②

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

ON A PROPOSED SYMBOLIC DYNAMICS FOR THE HENON MAP

by

Antonio Pietro Fontana

June 1993

Thesis Advisor: Jeffery J. Leader

REPORT DOCUMENTATION PAGE	*Form Approved* OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1993	Master's Thesis

4. TITLE AND SUBTITLE On a Proposed Symbolic Dynamics for the Hénon Map	5. FUNDING NUMBERS
6. AUTHOR(S) FONTANA, Antonio Pietro	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release, distribution is unlimited	

13. ABSTRACT *(Maximum 200 words)*

The utility of a computationally simple yet cryptologically robust rule for generating pseudorandom bitstreams cannot be overstated. In most applications we strive to detect and avoid chaotic behavior; here we embrace a particular chaotic discrete dynamical system to exploit its use as a driver for a pseudorandom number generator. The map from the Hénon attractor to the binary domain {0,1} proposed by Forré/Heyman has been tested cryptologically and statistically with mixed results. In this thesis we mathematically evaluate this symbolic dynamics scheme to investigate more rigorously its utility as a plausible pseudorandom number generator. Specifically, we demonstrate how the property of being one-to-one holds, but that the property of being onto does not.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Symbolic Dynamics, Chaos, Pseudorandom number generator, Hénon	169
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

On a Proposed Symbolic Dynamics for the Hénon Map

by

Antonio Pietro Fontana
Lieutenant , United States Navy
B.S., United States Merchant Marine Academy

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
June 1993

Author: Antonio Pietro Fontana
Antonio P. Fontana

Approved by: [illegible]
Jeffery J. Leader

Ismor Fischer
Ismor Fischer

Richard Franke
Richard Franke
Department of Mathematics

ABSTRACT

The utility of a computationally simple yet cryptologically robust rule for generating pseudorandom bitstreams cannot be overstated. In most applications we strive to detect and avoid chaotic behavior; here we embrace a particular chaotic discrete dynamical system to exploit its use as a driver for a pseudorandom number generator. The map from the Hénon attractor to the binary domain {0,1} proposed by Forré/Heyman has been tested cryptologically and statistically with mixed results. In this thesis we mathematically evaluate this symbolic dynamics scheme and investigate more rigorously its utility as a pseudorandom number generator. Specifically, we demonstrate how the property of being one-to-one holds, but that the property of being onto does not.

Accesion For		
NTIS CRA&I	☒	
DTIC TAB	☐	
Unannounced	☐	
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

TABLE OF CONTENTS

ACKOWLEDGEMENT

This thesis is dedicated to my thesis advisor and second reader, two men whose genuine love for mathematics has inspired me greatly, and to my wife, Kelly. Without their efforts this document would not have been possible.

I. INTRODUCTION

A. HÉNON'S DISCRETE DYNAMICAL SYSTEM

Michel Hénon (1931-   ) was a French mathematician and astronomer who was interested in modeling astronomical behavior. Hénon learned how the meteorologist-mathematician Edward N. Lorenz (1917-   ) had developed a three-dimensional attractor to model the complicated dynamics of thermal convection. An attractor is characterized by the following definition.

DEFINITION: Let V be a subset of R^m and $F: V \rightarrow R^m$, where $m = 1,2,3$. Let A be a subset of V. Then A is an **attractor** of F if:

i. A is a closed *invariant* subset of V under F. The set A in R^m is invariant under $F: V \rightarrow R^m$ if $F(A) \subseteq A$.

ii. There is a neighborhood U of A such that if v is in U, $F^n(v) \rightarrow A$ as $n \rightarrow \infty$, where F^n denotes the n-fold composition of F [Ref.1:p.147,200].

The Lorenz attractor is generated by a system of three differential equations and relies upon integration which does not lend itself to timely and accurate computer calculations. [Ref.2] Hénon sought to *build* a simple discrete dynamical system that would retain the qualitative properties of the continuous Lorenz system. His goal was achieved through study of the Poincaré section of the Lorenz equations. [Ref.3] Thus, the Hénon attractor is, to some extent, a two-dimensional version of the Lorenz attractor. The Hénon map is the following quadratic recurrence which maps the Euclidean plane into itself. [Ref.7]

$$H : R^2 \rightarrow R^2$$

$$x_{n+1} = 1 - ax_n^2 + y_n \qquad (1.1)$$

$$y_{n+1} = bx_n \qquad (1.2)$$

Each point in R^2 has a unique image and preimage under H. The same is true of the inverse of the Hénon map. The constants a and b are real parameters. We can write the map as a transformation in the following form:

$$H(x,y) = (1 - ax_n^2 + y_n, bx_n).$$

Understanding the dynamics of the map is easier if we decompose it into the following independent transformations:

$$H_1(x,y) = (x, y + 1 - ax^2)$$

$$H_2(x,y) = (bx, y)$$

$$H_3(x,y) = (y, x)$$

such that

$$H(x,y) = H_3(H_2(H_1(x,y))).$$

Respectively the transformations cause bending, contraction (for | b |<1), and a reflection. [Ref.5:p.662] This stretching and folding are analogous to the kneading or mixing common to chaotic systems.

The parameters (a,b) control whether the system forms an *interesting* attractor with apparent aperiodicity or an *uninteresting* attractor of low period. Reportedly, Hénon chose his **classical** parameters (a,b) = (1.4,0.3) because they gave the "nicest and strangest" picture. Figure 1 shows the computer representation of the Hénon attractor for $(x_0,y_0)=(0,0)$ using 2000 points.

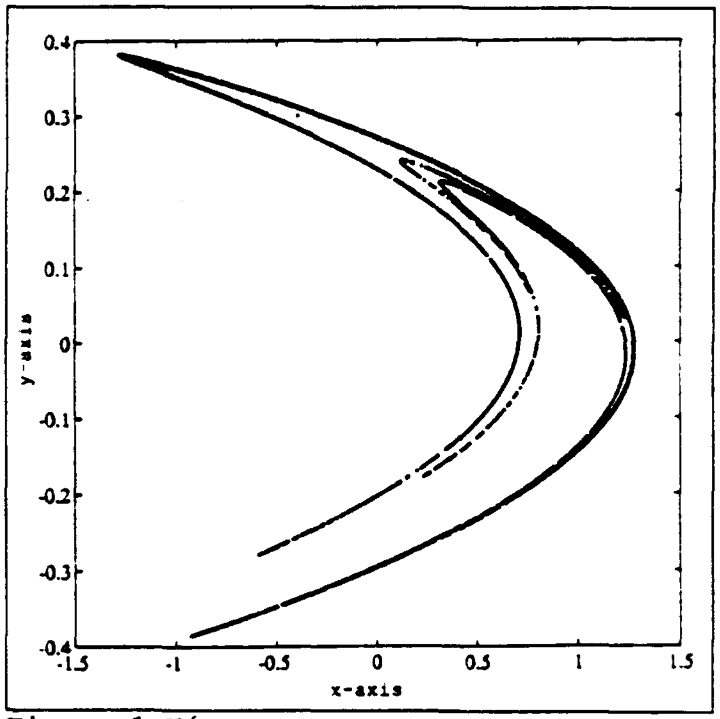

Figure 1 Hénon attractor

In keeping with our definition, the computer representation is generally unaltered by the choice of (x_0,y_0). This is true as long as the values tend to the attractor and the first, say, 100 iterates are ignored. The first points generated by the map may not be close to the attractor, but the generated points get closer and closer to the invariant set.

DEFINITION: Let x belong to the domain of a function f; x is of **period n** if $f^n(x) = x$, and is of **prime period n** if $x, f(x), f^2(x), \ldots, f^{n-1}(x)$ are distinct. If x has period n then the orbit $\{x, f(x), f^2(x), \ldots, f^{n-1}(x)\}$ is a **periodic orbit** [Ref.1:p.21]. If $n = 1$, then x is said to be a **fixed point**.

The merit of Hénon's choice of (a,b) parameters can be found by observation of the respective **bifurcation diagrams**. A bifurcation diagram gives us information regarding the periodic and nonperiodic orbits of a map as a parameter is altered. In a bifurcation diagram, the vertical axis represents iterates of the variable x as a parameter is varied on the horizontal axis. [Ref.1:p.52] If we take a vertical slice at a particular parameter we get an idea of the attractor's structure for that parameter. In Figure 2 the b-parameter is held fixed at 0.3 and the a-parameter is varied from 0.2 to 1.42.

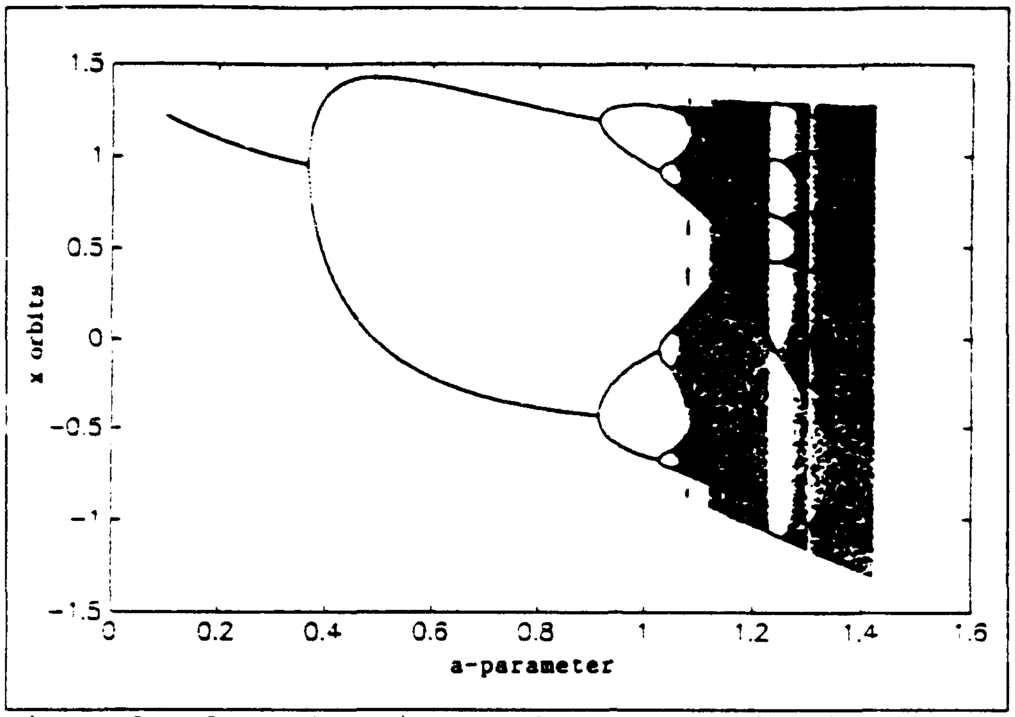

Figure 2 Bifurcation Diagram: b-parameter fixed

Notice that from about a = 0.1 to a = 0.4 we have a *point attractor*, that is, the entire attractor consists of a single fixed point. From about a = 0.4 to a = 0.9 we have a period-two attractor; that is, the attractor is made up of two points. These **period-doubling bifurcations** persist as the parameter a is increased, until a critical value of approximately a = 1.1 is reached, where we enter the *chaotic regime*. Notice that Hénon's classical a-parameter 1.4 lies in such an area of wildly complicated dynamics, sometimes called a *chaotic band* [Ref.6].

In Figure 3 the a-parameter is held fixed at 1.4 and the b-parameter is varied from 0.0 to 0.32. Again, Hénon's b-value 0.3 not surprisingly lies in an area of chaotic activity.

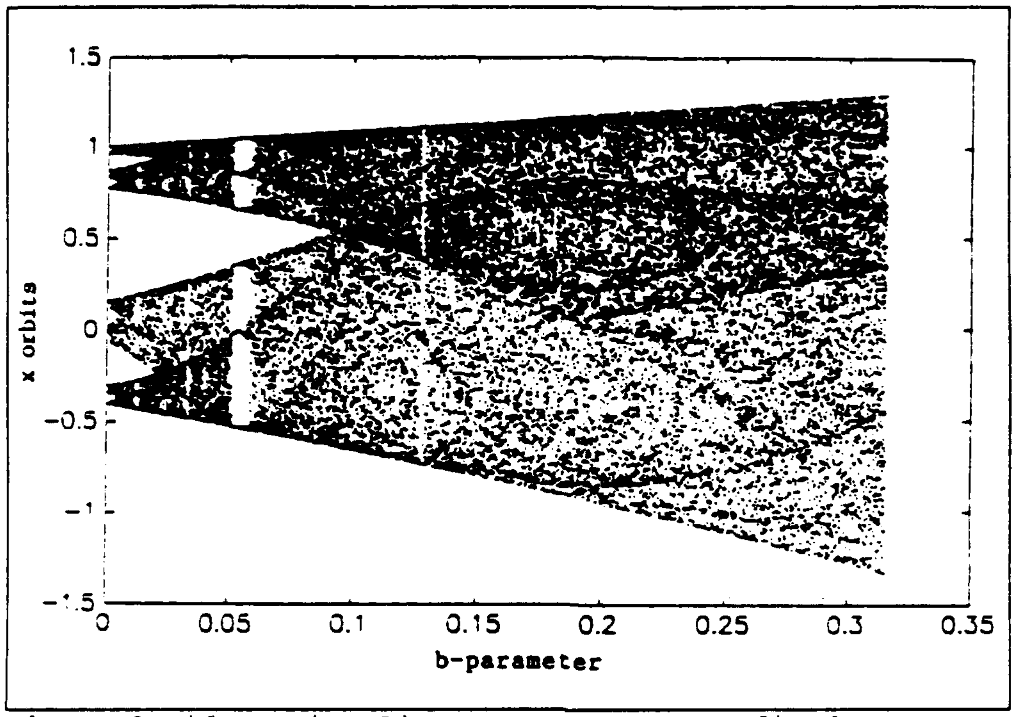

Figure 3 Bifurcation Diagram: a-parameter fixed

B. NATURE AND STRUCTURE OF THE HÉNON ATTRACTOR

Because the Hénon transformation is quadratic, it is possible that even relatively small (x_0, y_0) values can produce an iterated sequence, or orbit, that escapes to infinity. [Ref.4:p.664] The Hénon attractor has an associated **basin of attraction** of points (found numerically) which in forward iteration are drawn to the attractor. A point in the basin of attraction thus tends to the invariant set. The Hénon attractor also has an associated area known as the *trapping* *region*. The trapping region is a quadrilateral within the basin of attraction defined by the vertices:

$A = (-1.33, 0.42)$ $\quad$ $B = (1.32, 0.133)$

$C = (1.245, -0.14)$ $\quad$ $D = (-1.06, -0.5)$

If (x_0, y_0) is chosen from this region, all subsequent iterates will remain in the region [Ref.5:p.664]; hence the trapping region is invariant. Let

H_A = Hénon attractor

B_A = Basin of attraction

TR = Trapping region.

Then, we have that

$$H_A \subset TR \subset B_A.$$

This relationship is shown explicitly (excluding the trapping region) in Figure 4; the shaded region is the basin of attraction.

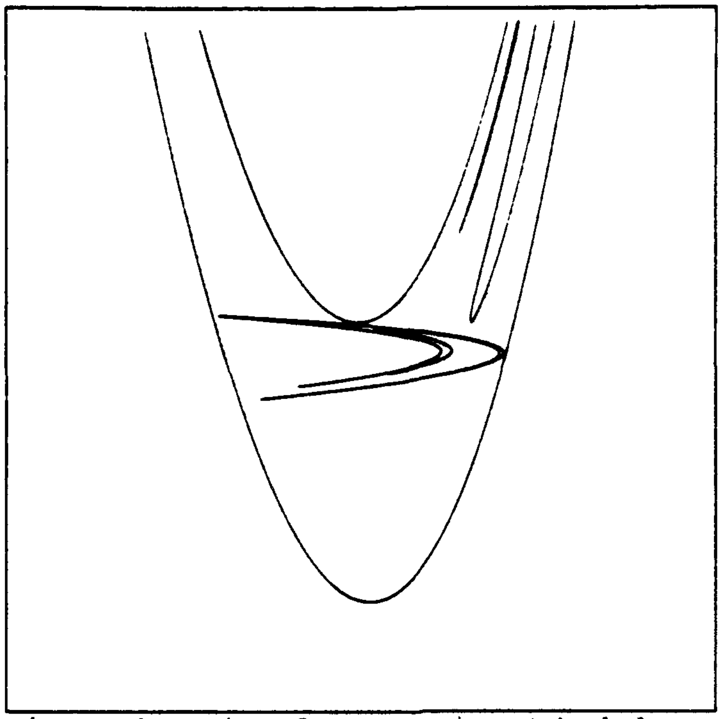
Figure 4 Basin of attraction (shaded region) and Hénon attractor. This figure is from reference 5, p.665.

Therefore,

i. if $(x_o, y_o) \in TR$, then $(x_n, y_n) \in TR$ for all $n > 0$.

ii. if $(x_o, y_o) \in B_A$ but $(x_o, y_o) \notin TR$, then (x_n, y_n) may not belong to TR for $n < N$. However, from the description of the basin of attraction we know that for all n sufficiently large, $(x_n, y_n) \in TR$.

The b-parameter, 0.3, is in the **dissipative term** of the relation. The Jacobian of the Hénon transformation is (-b), and hence the map is dissipative when $|b| < 1$. [Ref.1:p.168]. Therefore, should we iterate the entire trapping region under the transformation for b = 0.3, this area would *shrink* and in the limit tend to zero. The Hénon attractor thus has zero area. Shown symbolically,

$$H_A = \lim_{n \to \infty} H^n(TR)$$

or

$$H_A = \bigcap_{n=1}^{\infty} H^n(TR).$$

In fact, any point belonging to the trapping region can be iterated to construct a dense orbit on H_A, except the periodic points which are countable and hence have Lebesgue measure zero. The visual result, in the limit, is the same for any initial condition (less a finite number of initial iterations to assure close proximity to the attractor). Two different points will generate different sequences but the limit set, under the above qualification, will appear to be the same visually. Furthermore, in general, there is no correlation between the trajectories of two different initial points, except possibly in the first relatively few iterations should the initial points be neighbors. (We disregard here the instance where, say, trajectory 1 has common points with trajectory 2 but for different iterates. In this case the orbits are shifted versions of one another.) This lack of correlation is related to sensitive dependence on initial conditions.

The Hénon equations (1.1) and (1.2) (with the classical parameters (a,b)=(1.4,0.3)) may be written in one variable as:

$$x_{n+1} = 1 - 1.4x_n^2 + 0.3x_{n-1}. \qquad (1.3)$$

There are two fixed point solutions to the recurrence, where

$$x_{n+1} = x_n = x_{n-1}.$$

This occurs when (x,y) = (.6314...,.1894...) (FP2) and (x,y) = (-1.1314..., -.3393...) (FP1). The fixed points, trapping region and Hénon attractor appear in Figure 5.

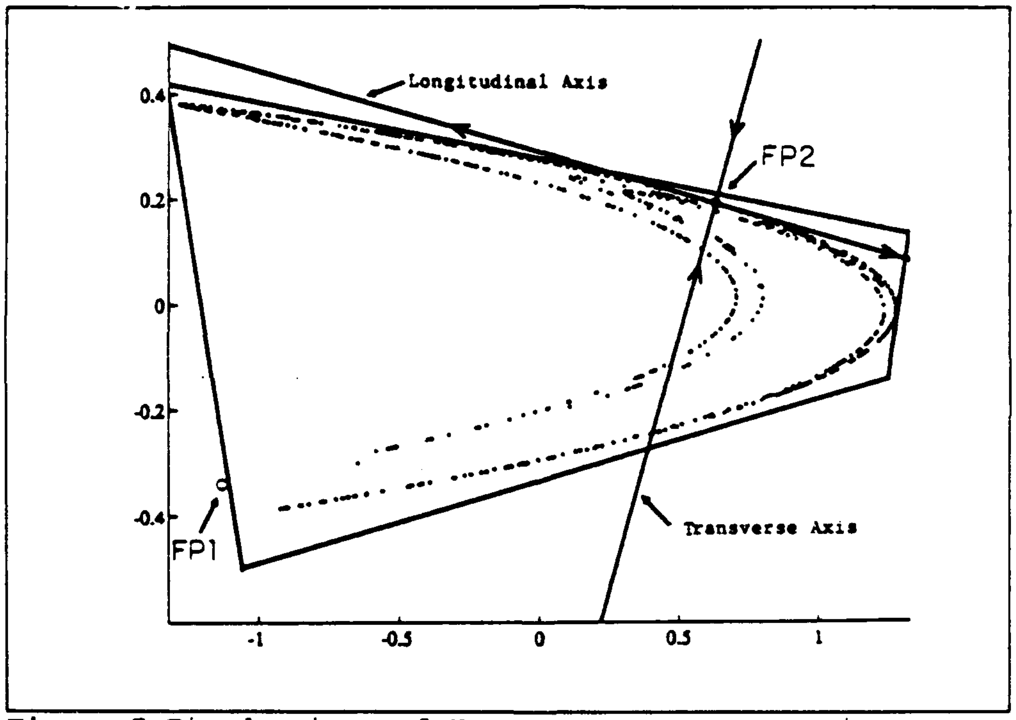

Figure 5 Fixed points of Henon attractor, trapping region and eigenvectors associated with FP2

The left fixed point FP1 is repelling and is noticeably not on the attractor or even within the trapping region. The remaining fixed point FP2 is a saddle point and belongs to the attractor. The Jacobian matrix at FP2 has associated eigenvalues λ_1 and λ_2, with slopes p_1 and p_2 for the respective eigenvectors:

$$\lambda_1 = 0.1559... \qquad p_1 = 1.9237...$$
$$\lambda_2 = -1.9237... \qquad p_2 = -0.1559...$$

Figure 5 also shows us the strong local effect of the eigenvalues at FP2. Longitudinal spreading is due to λ_2. Since $|\lambda_2|$ is greater than 1, points are forced outward from FP2 along the longitudinal axis forming a roughly broken line, at least locally. The eigenvalue λ_1 causes a transverse contraction of points. Figure 6 shows a detail of this structure.' The dense transverse structure is revealed through local magnification in Figure 7.

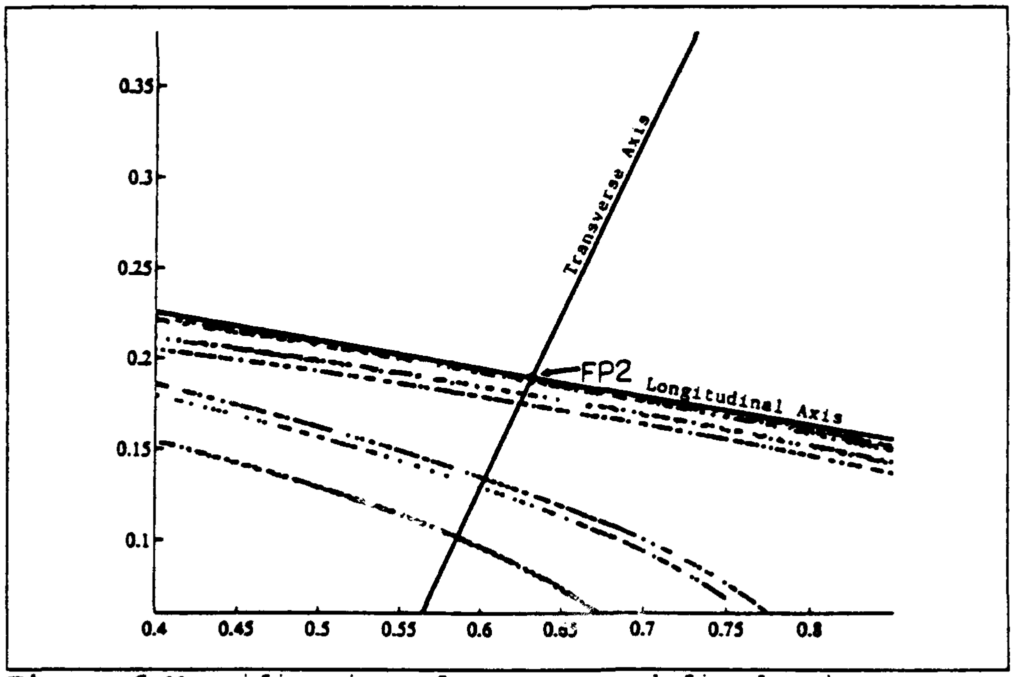

Figure 6 Magnification of area around fixed point

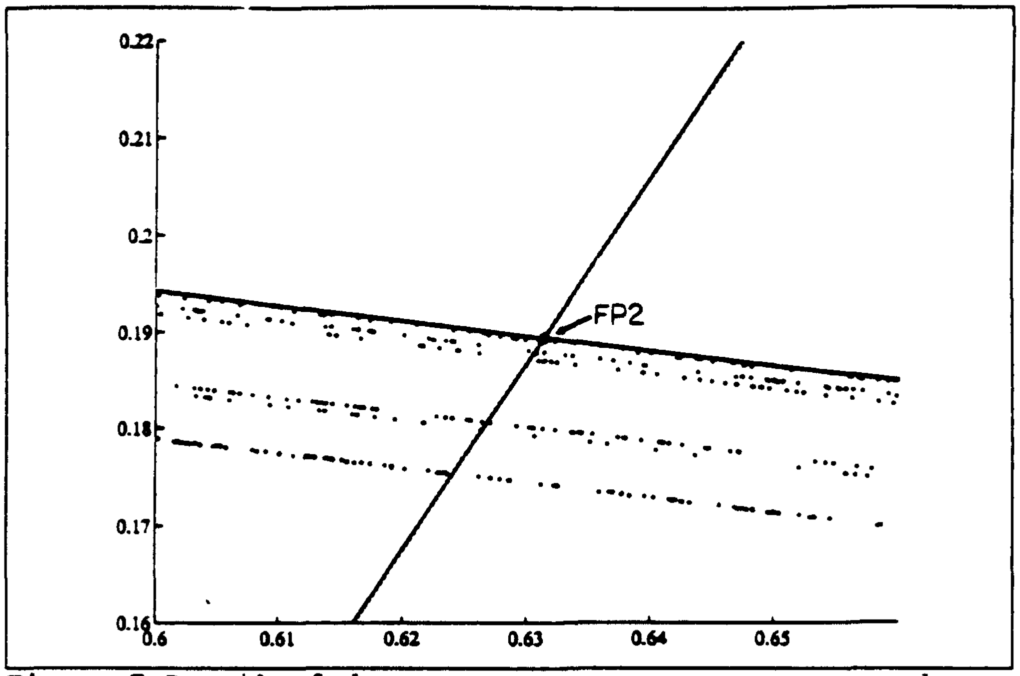

Figure 7 Detail of dense transverse structure around fixed point

As magnification is increased and more points are plotted the attractor begins to take on the appearance of distinct bands with a cross-section that has the nature of a one-dimensional Cantor set -- a totally disconnected set of points with patterns that persist on infinitely small scales. This is the *self-similar* nature of the attractor. Thus, at least on a level local to FP2 we observe a fractal quality that is common in chaotic attractors [Ref.5:p.670].

C. CHAOS

Because the field of chaos is in its relative infancy not all terms are universal. The most mathematically rigorous and widely accepted definition of chaos for discrete dynamical systems is the following:

DEFINITION: Let S be a set. A mapping $f: S \rightarrow S$ is said to be **chaotic** on S if

1. f has sensitive dependence on initial conditions,
2. f is topologically transitive,
3. the set of periodic points is dense in S. [Ref.8:p.50]

We will now elaborate on each of these concepts in a bit more detail starting with sensitive dependence on initial conditions.

DEFINITION: A mapping $f: S \rightarrow S$ has **sensitive dependence on initial conditions** if at every $x \in S$, there exists an $\varepsilon > 0$ such that for each $\delta > 0$, there is a y in S and a positive integer n such that

$$\|x-y\| < \delta \quad \text{and} \quad \|f^n(x) - f^n(y)\| > \varepsilon. \qquad \text{[Ref.1:p.84]}$$

Based on empirical evidence, the Hénon attractor is widely believed to possess sensitive dependence on initial conditions. [Ref.7] The *stretch-and-fold* phenomenon mentioned previously ensures that close points will not be neighbors after a finite number of iterations. Figure 8 shows a plot of the Euclidean distance between the orbits of two initial conditions.

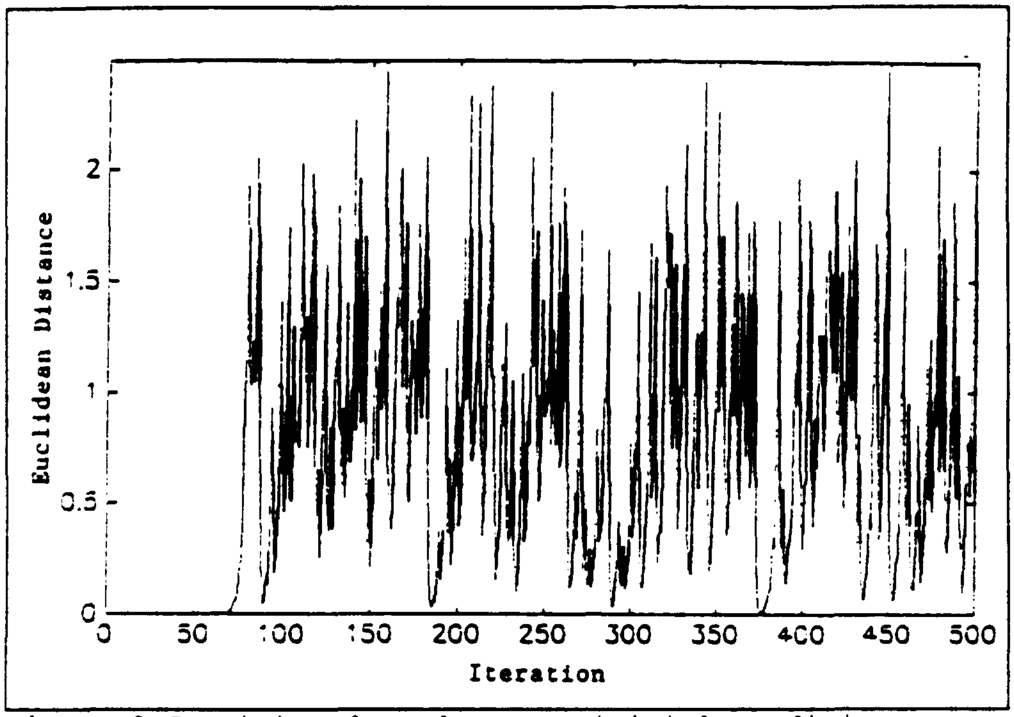

Figure 8 Sensitive dependence on initial conditions

The two points are the origin (0,0) and a point $(0,\varepsilon')$ where ε' is just within machine epsilon, that is $\varepsilon' < \varepsilon$. Machine epsilon in this case is approximately 2.22×10^{-16}. Within seventy iterations the orbits are completely different.

Of course, since both initial conditions belong to the trapping region the maximum difference between the orbits is bounded by the extremes of the trapping region. Although this is only the first of three criteria for chaos some accept the dynamics on the Hénon attractor as chaotic based on sensitive dependence alone. [Ref.4:p.671] Sensitive dependence brings out an interesting paradox. Globally we have an attractor with respect to the basin of attraction; however, locally for the points of the invariant set, the orbit jumps around the attractor chaotically, seemingly *repelled* by each point.

Topological transitivity, also called *mixing*, ensures that the set on which the dynamical system is defined cannot be divided into two disjoint open sets which do not "interact" (or more precisely, are invariant) under the system.

DEFINITION: A mapping $f: S \rightarrow S$ is **topologically transitive** if for any pair of open sets U, V, which are subsets of S, there exists $k > 0$ such that $f^k(U) \cap V \neq \emptyset$. [Ref.8:p.49]

Essentially, for a system to be topologically transitive it must be true that for any two arbitrarily small regions U and V (where U and V are not points) we can find an initial point x_0 in U whose orbit will enter the other region at some iteration. [Ref.5:p.554]

For a system to possess dense periodic points means that in an arbitrarily small neighborhood of any point belonging to the attractor there must be a periodic point. This condition suggests paradoxically that one of the qualities for "chaos" is a high degree of "order".

Although there exists vast numerical evidence to support the claim that Hénon's dynamical system possesses all three of these criteria for chaos, none of the criteria have been rigorously proven. [Ref.5:p.671] [Ref.19:p.152] To keep this in perspective, it must be acknowledged that our definition is more stringent than proving the existence of a dense orbit and a positive Lyapunov exponent, a commonly accepted alternative definition. [Ref.14]

D. THE SHADOWING LEMMA - TRUSTING THE COMPUTER

The study of theoretical chaotic dynamical systems would be extremely handicapped if we could not trust our computer representations of these systems. Because a computer is limited in terms of its accuracy by rounding error, it is appropriate for us to ask whether our computer calculation is an approximation of some true orbit of the system. [Ref.9] The Shadowing Lemma shows that we can trust the computer image.

DEFINITION 1: The map $f:R^m \rightarrow R$ is a **class C^p** map if it is p times continuously differentiable with respect to $x_1, x_2, \ldots, x_m$ for some p, $1 \le p \le \infty$. A mapping $f:R^m \rightarrow R^m$ is class C^p if each component f_i of f,(where i=1,...,m) is p times continuously differentiable [Ref.10:p.1-2].

DEFINITION 2: A mapping $f:R^m \rightarrow R^m$ is a **diffeomorphism** if it is one-to-one, onto and both f and f^{-1} are continuously differentiable maps; f is a C^p-diffeomorphism if both f and f^{-1} are C^p maps [Ref.10:p.2].

DEFINITION 3: Let $f: R^2 \rightarrow R^2$ be a diffeomorphism. A set Λ is said to be a **hyperbolic set** under f if:

1. for all points p belonging to Λ, there is a set of lines $E^s(p)$ and $E^u(p)$ in the tangent plane at p which are preserved by the Jacobian of f at p. ($E^s(p)$ is called the stable line and $E^u(p)$ is called the unstable line).

2. $E^s(p)$ and $E^u(p)$ vary continuously with p.

3. There is a constant $\lambda>1$ such that $|J(p)(v)|\geq\lambda|v|$ for all $v\in E^u(p)$ and $|J^{-1}(p)(v)|\geq\lambda|v|$ for all $v\in E^s(p)$. [Ref.8:p.236-237]

Suppose f satisfies the three previous definitions for some p with an associated hyperbolic invariant set Λ. The true (forward) orbit of a point z_0 under the map f is given by the sequence $\{z_i\}_0^\infty$ such that $z_i = f^i(z_0)$. [Ref.10:p.351] Due to finite precision the exact calculation of the sequence $\{f^i(z_0)\}_0^\infty$ is impossible. Because we are unable to obey the function $z_i = f^i(z_0)$ in order to find an exact orbit, let us instead define a *pseudo-orbit* or more specifically ε-pseudo-orbit.

DEFINITION 4: An **ε-pseudo-orbit** is a sequence of points, $\{y_i\}_0^\infty$, such that $y_i \in \Lambda$, and

$$d(y_{i+1}, f(y_i)) < \varepsilon$$

where d is a metric on R^n. [Ref.10:p.352] The ε-pseudo-orbit is pictured at the top of Figure 9.

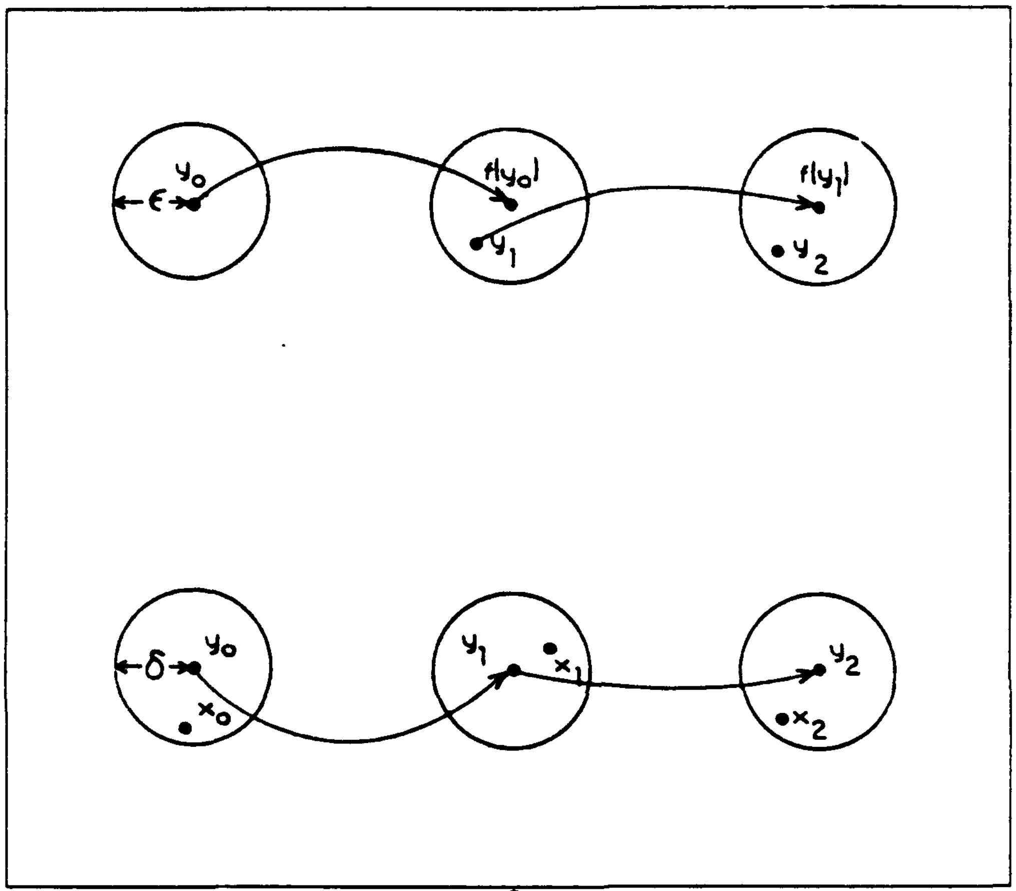
y_0
$\leftarrow \epsilon \rightarrow$
$f(y_0)$
y_1
$f(y_1)$
y_2
$\leftarrow \delta \rightarrow$
y_0
x_0
y_1
x_1
y_2
x_2

Figure 9 ε-Pseudo-orbit and δ-Shadow

Notice that we adjust each iterate by dropping $f(y_i)$ and then choosing y_{i+1} to be within ε of $f(y_i)$. We can see that the term *pseudo-orbit*, therefore, is appropriate. One final definition is necessary for us to understand the Shadowing Lemma.

DEFINITION 5: Suppose $\{x_i\}_{i=0}^{\infty}$ is an actual orbit, that is, $x \in \Lambda$ and $x_i = f^i(x_0)$; $\{x_i\}_{i=0}^{\infty}$ is a **δ-shadow** of the pseudo-orbit $\{y_i\}^{\infty}$ if

$$d(x_i, y_i) < \delta.$$

The δ-shadow of the pseudo-orbit is shown at the bottom of Figure 9. The Shadowing Lemma assures us that within an arbitrarily small distance, δ, of an ε-pseudo-orbit there is an exact orbit. Stated succinctly:

Shadowing Lemma: If Definition 2 holds for the map f and Λ is a hyperbolic invariant set on R^n, then for every $\delta > 0$ there is an $\varepsilon > 0$ such that every ε-pseudo-orbit in Λ is δ-shadowed by the actual orbit of some point $x \in \Lambda$. [Ref.10:p.352]

Although the Shadowing Lemma is very powerful, it does not explicitly speak to us about the computed orbit, call it w_i, for which we hope to find to an exact orbit nearby.

Our computed orbit, w_i, is not the true orbit $\{f^i(y_0)\}^{\infty}$ we are looking for; however, let us show more clearly why it is an acceptable approximation of a true orbit in the system. [Ref.9:p.251] The sequence of computed values, w_i, from the Hénon map for some initial condition y_0 is actually the floating point representation of $f(w_{i-1})$ or,

$$
\begin{aligned}
w_0 &= y_0 \\
w_1 &= fl(f(y_0)) \\
w_2 &= fl(f(w_1)) \\
w_3 &= fl(f(w_2)) \\
&\vdots \\
w_{n+1} &= fl(f(w_n)).
\end{aligned}
$$

Recall from our definition that $f(y_i)$ is within ϵ of y_{i+1}. In order for us to show that a computed orbit, w_i, can be arbitrarily close to an exact orbit, x_i, which δ-shadows the orbit, y_i, we must first make an important assumption. We assume that at each iteration the error between the computed value of $f(y_i)$ and the exact value of $f(y_i)$ is bounded. That is,

$$d(w_i, f(y_{i-1})) < \epsilon' \qquad \text{such that } \epsilon' < \epsilon_r$$

where ϵ_r is some function of the machine precision which bounds the error for all iterations. Figure 10 shows a representative configuration of the orbits involved. By the Shadowing Lemma $d(y_i, x_i) < \delta$.

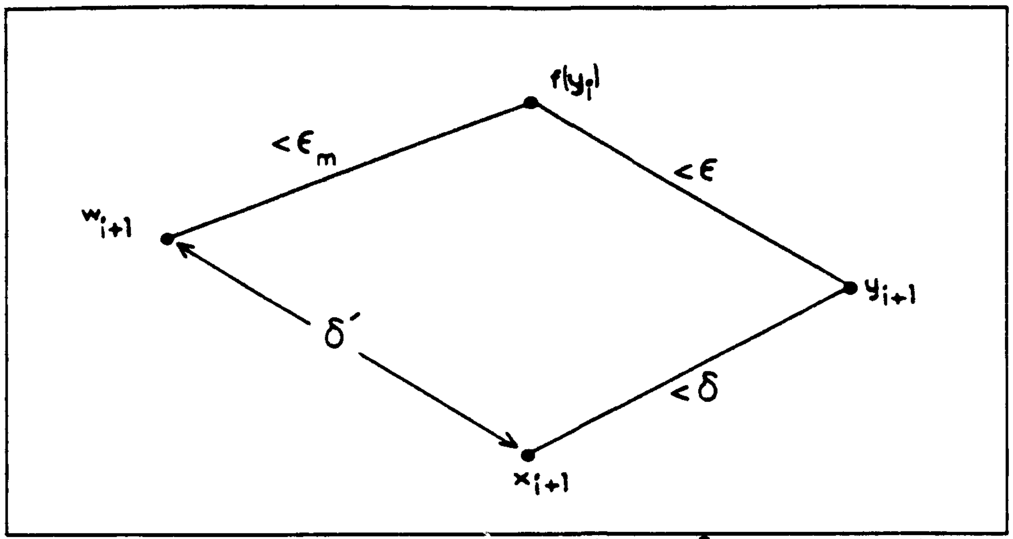

Figure 10 Pseudo-orbit, exact orbit in δ-shadow, and computed orbit

Using the triangle inequality it follows that

$$d(w_i,x_i) \le d(x_i,f(y_{i-1})) + d(w_i,f(y_{i-1})).$$

But,

$$d(x_i,f(y_{i-1})) \le d(x_i,y_i) + d(y_i,f(y_{i-1})).$$

Therefore,

$$d(w_i,x_i) \le d(x_i,y_i) + d(y_i,f(y_{i-1})) + d(w_i,f(y_{i-1}))$$

or

$$d(w_i,x_i) \le \delta + \varepsilon + \varepsilon_m.$$

If we let $\delta' = \delta + \varepsilon + \varepsilon_m$ then it follows that, for every $\delta' > \varepsilon_m$ we can calculate a computed orbit, w_i, that is δ'-shadowed by an exact orbit, x_i. In this way we see that the existence of our computed orbit in the δ'-shadow of an exact orbit actually depends on the precision of the machine in use.

According to a theorem in a recent paper (Ref.15), specific criteria have been developed under which we are assured that in the shadow of a computed orbit there will always exist an exact orbit within a *calculable* tolerance. The authors accepted that the Hénon map and the Hénon attractor satisfied Definitions 1 and 2 previously mentioned. The accuracy of the numerically computed orbits which the theorem measures applies to non-hyperbolic sets as well as hyperbolic sets so compliance to Definition 3 may be relaxed [Ref.17]. Computations applying the theorem in reference 15 were performed in Microsoft Quickbasic using a double precision IBM system computer. The value of ε (the distance between y_n and $f(y_{n-1})$ which defines the pseudo-orbit) was held to approximately 2^{-46}. The authors applied the theorem and it was discovered that even after 372,000 iterations there is a true orbit which differs by at most 2^{-29} (approximately 1.8626×10^{-9}) from the computed orbit generated by the Hénon map.

Based on the dense structure and infinite detail of the Hénon attractor it is intuitively reasonable that in the *foot-print* of the computed orbit there exists an exact orbit. Thus, the Shadowing Lemma assures us that the statistical evidence measured under computer analysis is significant.

II. PSEUDORANDOMNESS FROM CHAOS

A. LINKING CHAOS TO THE BINARY DOMAIN

Symbolic dynamics is a technique that can be used to relate the dynamics of a particular system on a metric space to another system on symbol space. Symbolic dynamics allows us to analyze a system by studying its effect on symbol space. The principal idea is that if two systems are *topologically conjugate*, then their dynamics are **equivalent**.

DEFINITION: Suppose we have two maps: $f: U \rightarrow U$ and $g: V \rightarrow V$. The functions f and g are **topologically conjugate** if there exists a *homeomorphism* $h: U \rightarrow V$ such that $h \circ f = g \circ h$. The function h is a homeomorphism if:

i. h is one-to-one

DEFINITION: Suppose f is a function from U to V; f is **one-to-one** if and only if for all elements u_1 and $u_2 \in U$, $f(u_1) = f(u_2)$ implies $u_1 = u_2$.

ii. h is onto

DEFINITION: Suppose f is a function from U to V; f is **onto** if and only if for any element v in V there exists an element u in U such that $v = f(u)$.

iii. h is continuous

iv. h^{-1} is continuous.

Criterion iv is actually implied by criteria i, ii, and iii. Furthermore, if f is topologically conjugate to g then it follows that g is topologically conjugate to f.

Finding a homeomorphism that forms a topological conjugacy between two maps can be extremely difficult. However, this does not diminish the power of the relation. A topological conjugacy, as we mentioned, relates the dynamics of two systems completely. [Ref.20:p.27] For example, if the function f, from our definition, has a period-two cycle {p,q}, then {h(p),h(q)} is a period-two cycle for the function g. In this way, all orbits for the function f have corresponding orbits for the function g. Furthermore, if f has a dense set of periodic points in U, then the same is true for g in V. [Ref.5:p.571] Following this reasoning, if it can be shown that the dynamics of map f exhibit the three characteristics of chaos previously mentioned, and there exists a topological conjugacy between f and g, then map g will also be chaotic. [Ref.20:p.28]

Can we harness this *chaotic energy* that exists in one system and through an effective symbolic dynamics transfer it to another under a homeomorphism to achieve a useful result? This is precisely the issue raised in Réjàne Forré's treatise of November 1990. [Ref.12] Forré hoped to apply a presumably chaotic discrete dynamical system to the field of cryptography. Specifically, she attempted to devise a scheme to generate nearly random sequences of zeros and ones which could be used for coding purposes. (Any numbers generated by a deterministic rule cannot be truly random and are, therefore, termed *pseudorandom.*) Forré's proposed symbolic dynamics relates the apparent chaotic dynamics of an orbit on the Hénon map to binary codespace by

$$h: H_{\Lambda} \rightarrow \Sigma_2 .$$

Here, Λ represents the attractor associated with the mapping f (see definition Chapter I. section A. p.1); in particular, f is the Hénon map. The set Σ_2, also called symbol space or codespace, represents the collection of all infinite sequences of zeros and ones. Any chaotic behavior exhibited in the Hénon map transferred to Σ_2 would be observed as a pseudorandom stream of zeros and ones in Σ_2. Forré's symbolic dynamics is based solely on the horizontal component, x_i, of the iterates of the Hénon map. The elements of such a binary sequence $\{S_i\}_{i=1}^{N}$ in Σ_2 are defined as follows:

$$\text{if } x_i \le x_{MED}, \text{ then } S_i = 0;$$

$$\text{if } x_i > x_{MED}, \text{ then } S_i = 1,$$

for a string of length N where x_{MED} represents a *dynamic median*. (The dynamic median ensures that on the average the trajectory will fall on each side of the median for half the iterates; this median has been more carefully calculated as $x_{MED}=.4098$ in reference 13). In order to clarify the computational process involved in Forré's symbolic dynamics the following example is furnished. Let us arbitrarily choose a particular (x_0,y_0) value from the trapping region, say $(x_0,y_0) = (1,0)$. Since $x_0 > x_{MED}$ the binary sequence element corresponding to (x_0,y_0) is $S_0 = 1$. Using equations (3.1) and (3.2) below we calculate (x_1,y_1) as follows:

$$x_{n+1} = 1 - 1.4\ x_n^2 + y_n \qquad (3.1)$$

$$y_{n+1} = .3x_n \qquad (3.2)$$

$$x_1 = 1 - 1.4(1)^2 + 0 = -.4$$

$$y_1 = .3(1) \qquad = \quad .3$$

Because $x_1 \leq x_{MED}$ the next binary sequence element, corresponding to (x_1,y_1), is $S_1 = 0$. In this way we can calculate the entire forward sequence of binary elements:

$$S = \{S_0,S_1,S_2,S_3...\} = \{1,0,1,0,...\}.$$

Although Forré's calculated median was inaccurate (she used a dynamic median $x_{MED} = .39912$) her results showed that under the cryptographic properties of linear complexity and jump complexity the bitstreams were *wholly indiscernible* from truly random sequences. However, she concluded that a third property, the *n-tuple distribution*, was inconsistent with truly random sequences and that this rendered the unaltered scheme unsuitable as a pseudorandom number generator.

B. RECENT STUDY OF FORRE'S SYMBOLIC DYNAMICS

Forré drew attention to the poor n-tuple characteristics of her unaltered symbolic dynamics. In a more recent study [Ref.13] Heyman refined the calculation of the median, x_{MED}, and rigorously investigated the n-tuple property, as well as three other cryptographic properties, to evaluate the pseudorandom number generator. It was concluded that the n-tuple or *runs* property was a minor detractor and that the proposed use of the scheme as a cryptographic pseudorandom number generator was *sound and effective*. The runs property was mentioned by both Forré and Heyman, but the authors' conclusions with regard to the significance of the property were markedly different. Under these mixed results, the *runs anomaly* demands further investigation.

Due to the disparate conclusions drawn by Forré and Heyman, it is a natural next step to evaluate the symbolic dynamics under more mathematically rigorous criteria in order to determine more objectively and conclusively whether the scheme is an effective pseudorandom number generator. Because the chaotic behavior on the attractor and certainly the symbolic dynamics scheme itself are mathematical concepts, we evaluate the pseudorandom number generator mathematically.

This is certainly necessary since the previous papers were not analytical and gave only a cursory mention of the theory of chaos. Primarily, our task is to collect evidence to prove or disprove that this scheme,

$$h:H_A \to \Sigma_L$$

gives a homeomorphism. Our secondary objective is to explain completely the runs anomaly in mathematical terms. Thus, based on the definition of a homeomorphism we will provide experimental evidence to support the one-to-one property. Furthermore, we will provide analytic proof that the presumed homeomorphism is not onto for the proposed pseudorandom number generator.

III. EXPERIMENTAL EVIDENCE FOR ONE-TO-ONE

A. A COMPUTER PARADIGM

The Hénon attractor $\Lambda = H_A$ is an infinite set of points produced by the Hénon map. The initial condition of the Hénon recurrence dictates which orbit on H_A we produce. Let λ_1, λ_2 be two infinitely long sequences of points (orbits) based on two different initial conditions (s_0, t_0) and (l_0, m_0) respectively, where (s_0, t_0), (l_0, m_0) are from H_A. The sequence λ_1 will be defined as $\{(s_0,t_0), (s_1,t_1), (s_2,t_2), \ldots\}$. The sequence λ_2 will be $\{(l_0,m_0),(l_1,m_1),(l_2,m_2),\ldots\}$. Although the two-dimensional plots of λ_1 and λ_2 are indistinguishable, the two sequences of points λ_1, λ_2 are not the same. Let $h(\lambda_1), h(\lambda_2)$ belong to Σ_2, the space of all infinite sequences of zeros and ones. For the symbolic dynamics $h: H_A \rightarrow \Sigma_2$ to be one-to-one, it must be true that if $h(\lambda_1) = h(\lambda_2)$ then $\lambda_1 = \lambda_2$. That is, if two binary sequences belonging to Σ_2 are identical then they must be mapped from the same initial point (x,y) in H_A.

We cannot hope to find with any precision the points belonging to H_A. Furthermore, by increasing the number of possible initial conditions or *keys* from which we can generate orbits close to the attractor we enhance the cryptographic qualities of the pseudorandom number generator. That is, we reduce substantially the possibility of the key being found and the code being broken. For these reasons we modify our symbolic dynamics scheme to $h{:}TR \rightarrow \Sigma_c$. That is, we increase the number of possible *keys* to all those possible points from the trapping region. Under the Hénon map the trapping region is also invariant; therefore, $h{:}TR \rightarrow \Sigma_c$ is a symbolic dynamics analogous to $h{:}H_A \rightarrow \Sigma_c$.

If two binary sequences are not the same then they must have originated from different initial (x,y) values. The total count of computer representable numbers in the trapping region, although extremely large, is finite. Thus the task of proving that the homeomorphism is one-to-one on an uncountably infinite number of points in the plane is avoided and unnecessary. Certainly, no irrational numbers are computer representable. Since all pseudorandom number generators are implemented in a computer environment we qualify our goal to showing that the presumed homeomorphism is *computer one-to-one* on TR. That is, we attempt to show that the homeomorphism is one-to-one with respect to the computer representable points in the quadrilateral.

We propose then a computer paradigm that models the finite number of computer representable numbers. The model will also give us insight into the complicated dynamics of the map. Consider the trapping region or *quadrilateral* as a grid consisting of a large but finite number of points. Figure 11 shows the quadrilateral and the median $x_{MED} = .4098$.

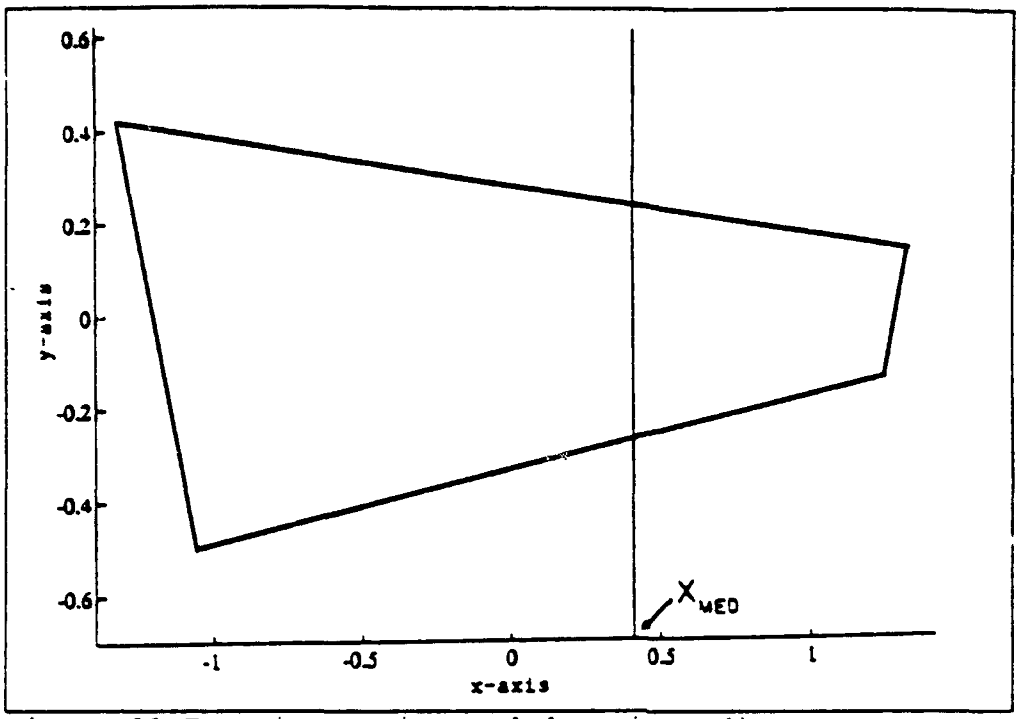

Figure 11 Trapping region and dynamic median

The quadrilateral is divided into two sides by the median. An initial point chosen from the *left quadrilateral* will correspond to a sequence under $h: TR \rightarrow \Sigma_2$ that will begin with the binary digit zero (0). Similarly, an initial point from the *right quadrilateral* will correspond to a sequence that will begin with the binary digit one (1). Let us focus on the right quadrilateral. (Our argument will extend to the left quadrilateral.)

We model the computer points in the right quadrilateral region by first enclosing it in a rectangle of minimum area. By choosing a particular *spacing* we can *fill* the rectangle with equally spaced points to produce a Cartesian coordinate system. If we disregard the points in the rectangle but outside the trapping region we have a *grid* which can be used to model the large but finite number of computer representable points. If we can show conclusively that as our spacing becomes smaller and smaller, the binary sequences of these points are *all* different from that of a point chosen at random not belonging to the grid points, then we should be convinced that h is one-to-one. Fundamentally, we first choose a fine grid spacing for the right quadrilateral that corresponds to a large number of points in the plane. As we iterate *all* the grid points *and* our random point through the Hénon map we cull out those points that at each iterate do not give the same binary element from {0,1} as the random point. We must convince ourselves that there exists some iterate for which all the binary sequences corresponding to the grid points differ from the binary sequence of the random point. Thus, we focus on a succession of subsets of our original grid points at each iteration. In this systematic way with computer assistance we hope to reveal the answer to our question.

B. PUTTING THE MODEL TO WORK

Let us begin with an example of our model in practice. This example was chosen because it allows the reader a clear, typical depiction of the procedure in a small number of iterations. Various MATLAB programs for this process on both sides of the quadrilateral can be found in Appendix A under the names GRDCOMP1.M through GRDCOMP4.M. As shown in Figure

12 the model lets us fill the right side of the quadrilateral with equally spaced points determined by a specified value for the spacing.

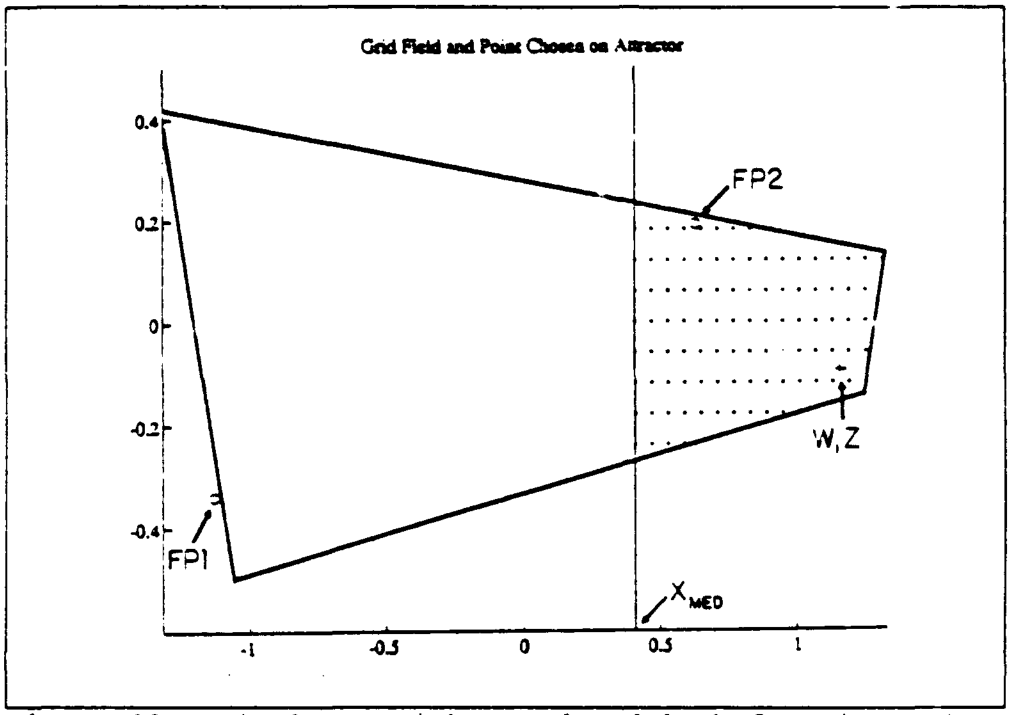

Figure 12 Typical run right quad model: before iteration

Here we use a relatively coarse spacing of 0.06. The point (W,Z) represented by a cross within the field is chosen at random but should be different from the grid points. Figures 13 through 21 represent subsequent iterates of the Hénon map where only those points from the grid which give the same binary sequence as the randomly chosen point (W,Z) are preserved. The median, x_{MED} = .4098, is the dynamic median, as previously mentioned; therefore, it is not surprising that we lose approximately half the points at each iteration.

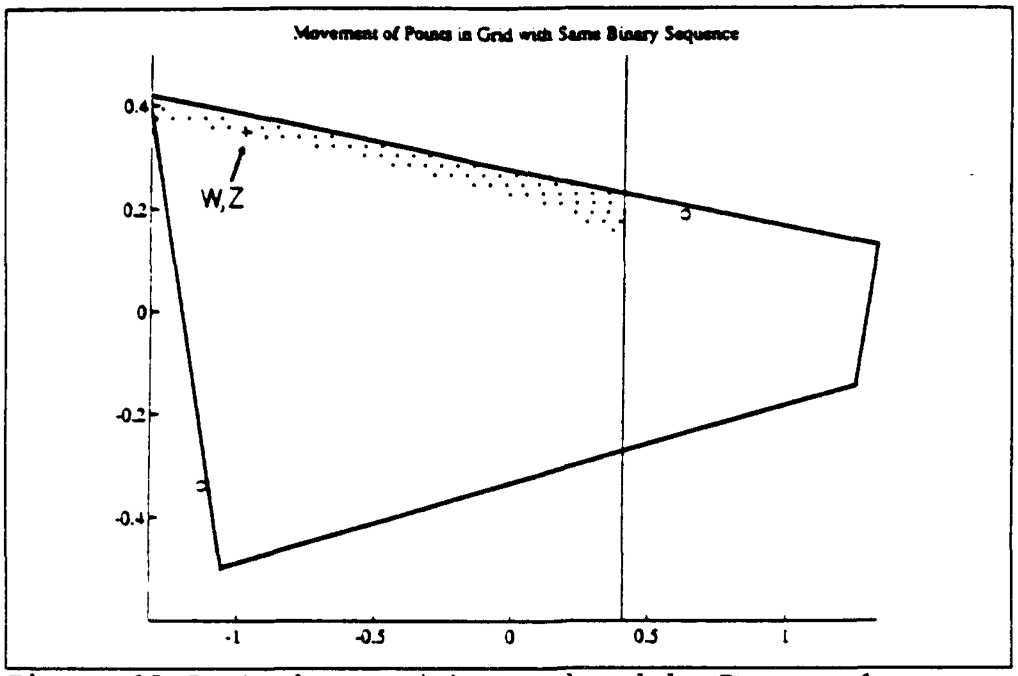

Figure 13 Typical run right quad model: Iterate 1

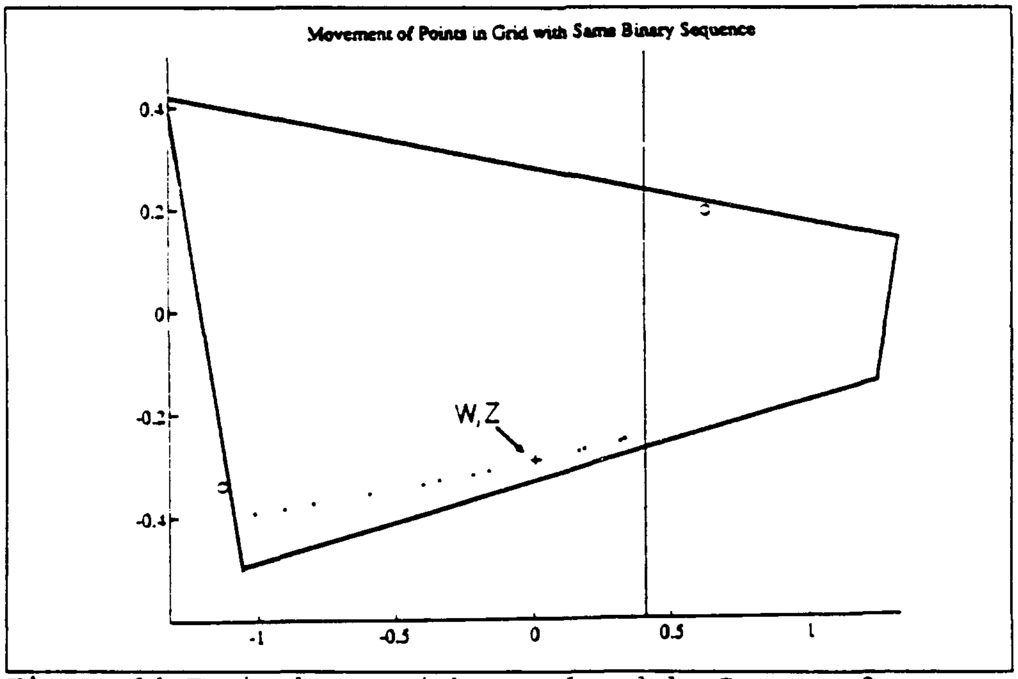

Figure 14 Typical run right quad model: Iterate 2

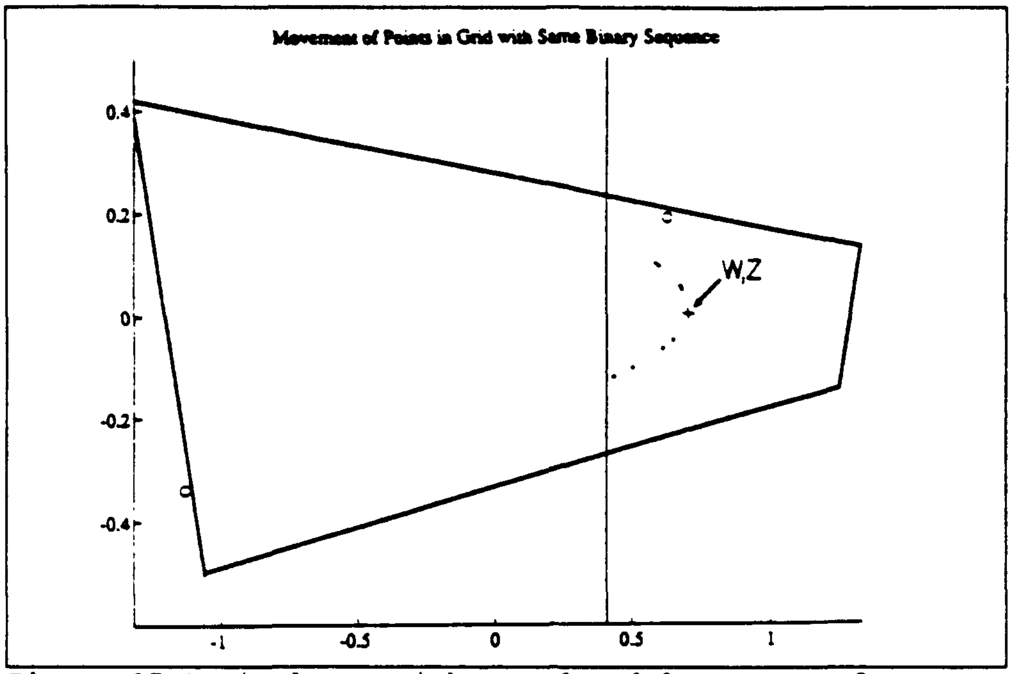

Figure 15 Typical run right quad model: Iterate 3

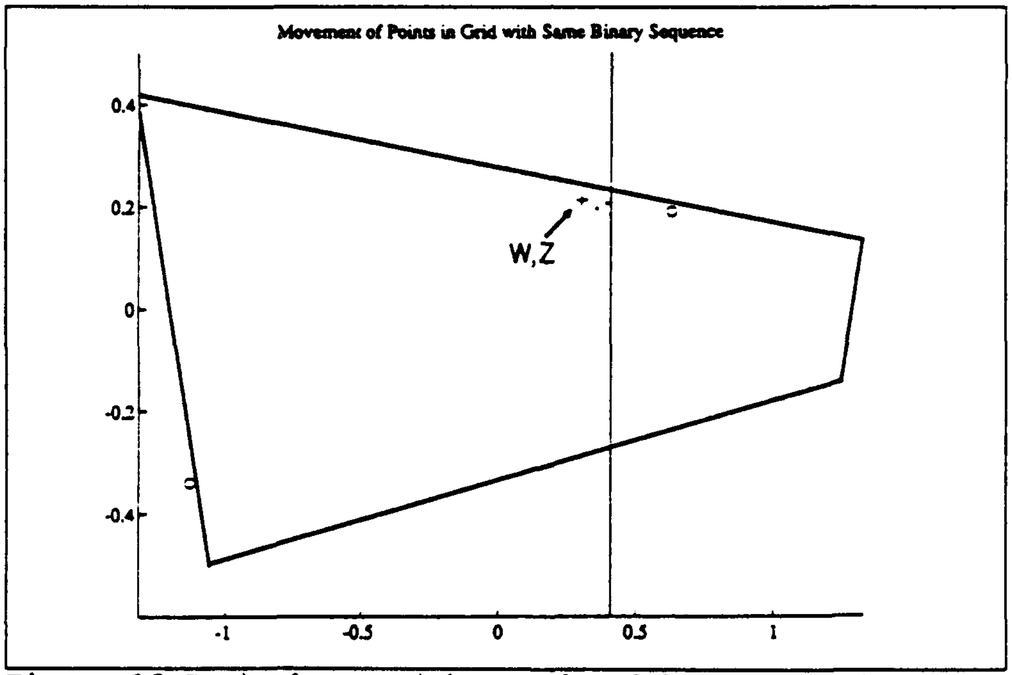

Figure 16 Typical run right quad model: Iterate 4

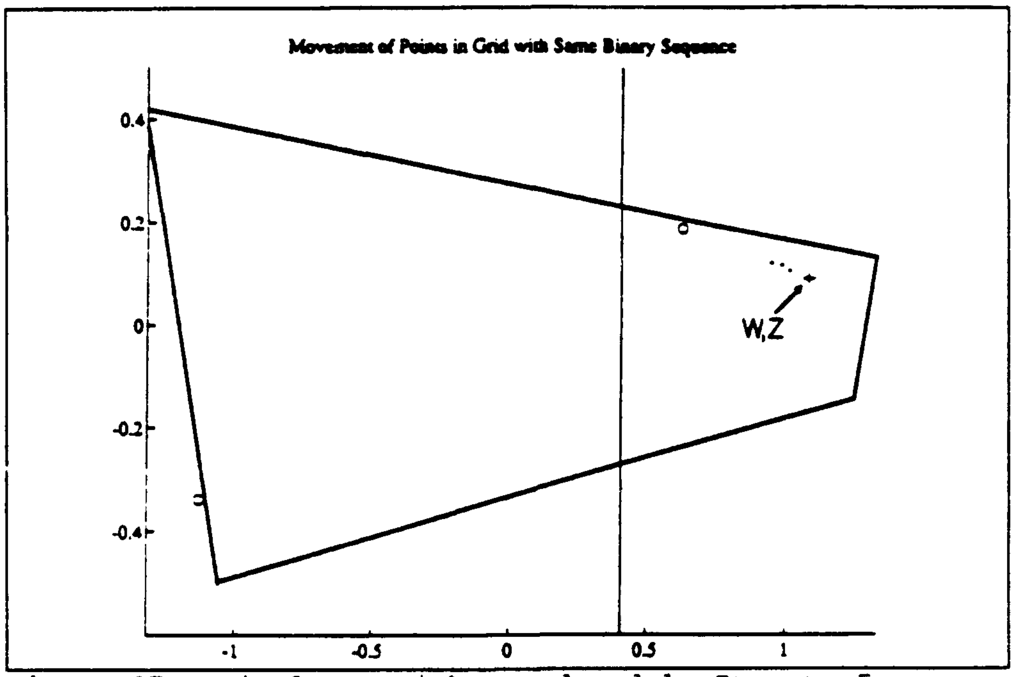

Figure 17 Typical run right quad model: Iterate 5

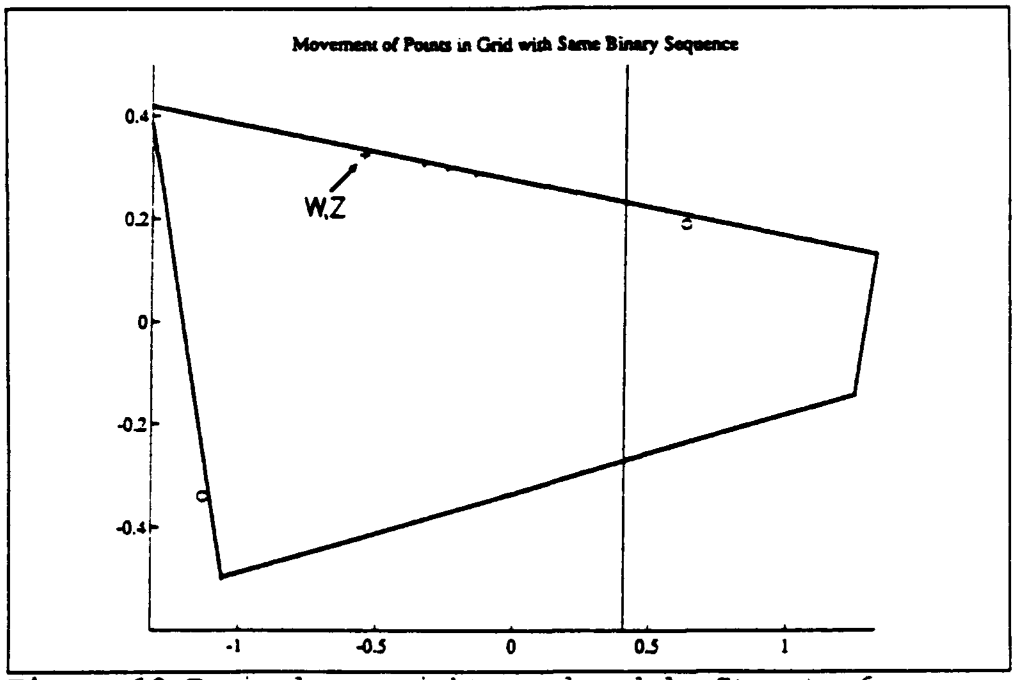

Figure 18 Typical run right quad model: Iterate 6

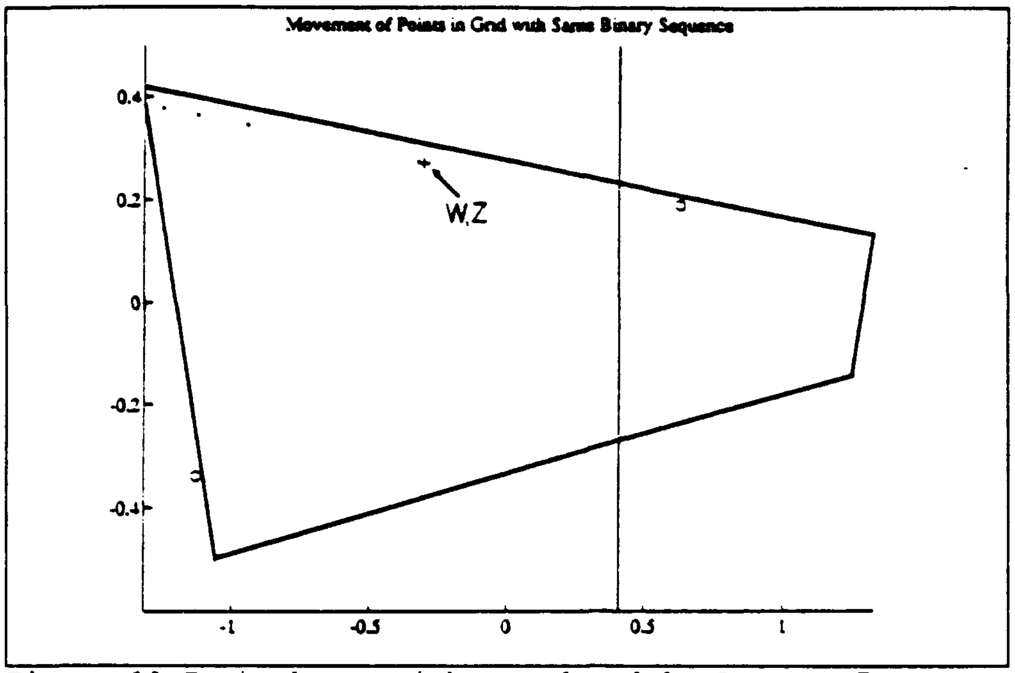

Figure 19 Typical run right quad model: Iterate 7

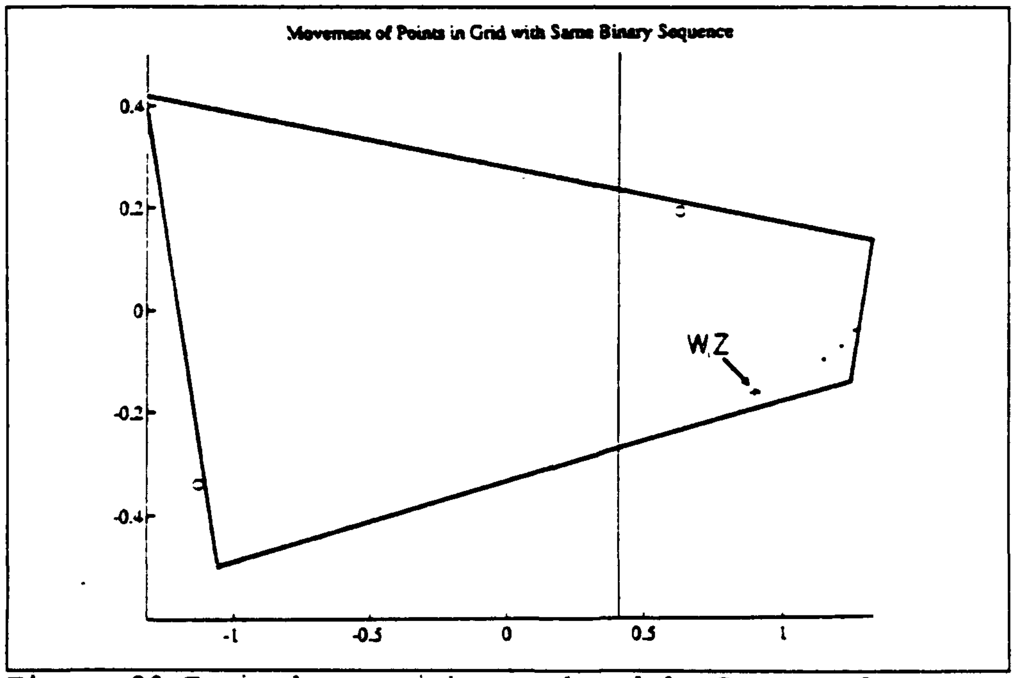

Figure 20 Typical run right quad model: Iterate 8

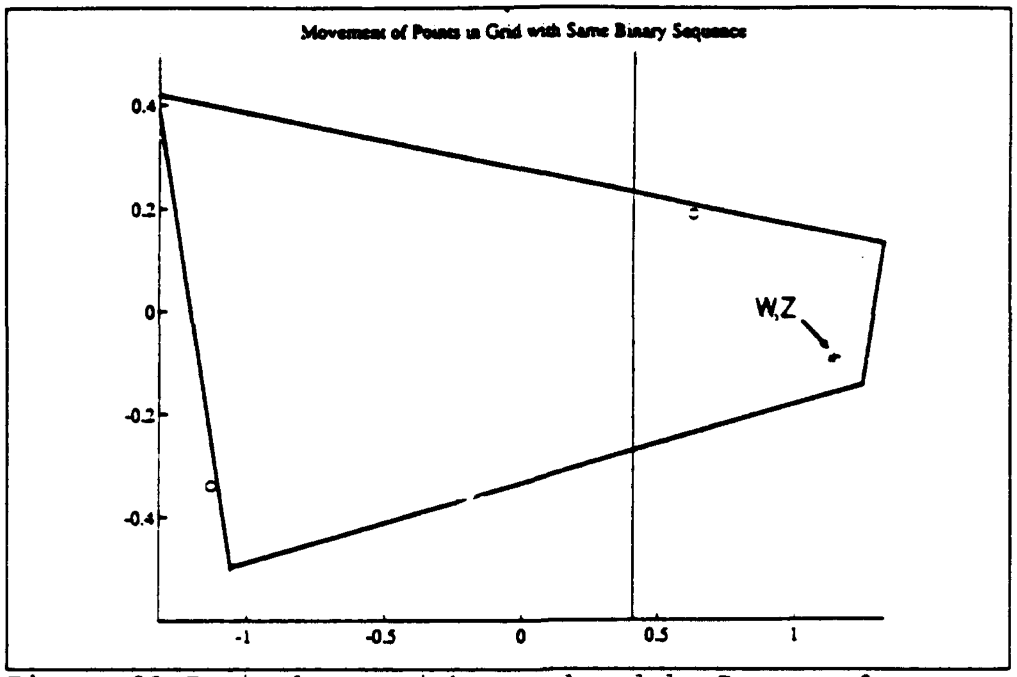

Figure 21 Typical run right quad model: Iterate 9

By iterate 10 (not shown) no grid point is on the same side of x_{MED} as (W,Z). Figure 22 gives us an idea of how quickly the points that *follow* the orbit of (W,Z) diminish in number. The curve $(1/2)^n$ is plotted as a comparison. TABLE 1 shows the count of points for iterates 1 through 9 that give the same binary sequence as (W,Z).

TABLE 1 NUMBER OF POINTS THAT GIVE THE SAME BINARY SEQUENCE

NUMBER OF POINTS THAT GIVE THE SAME BINARY SEQUENCE										
ITERATE	0	1	2	3	4	5	6	7	8	9
NO. PTS	98	65	13	9	4	4	4	4	4	1

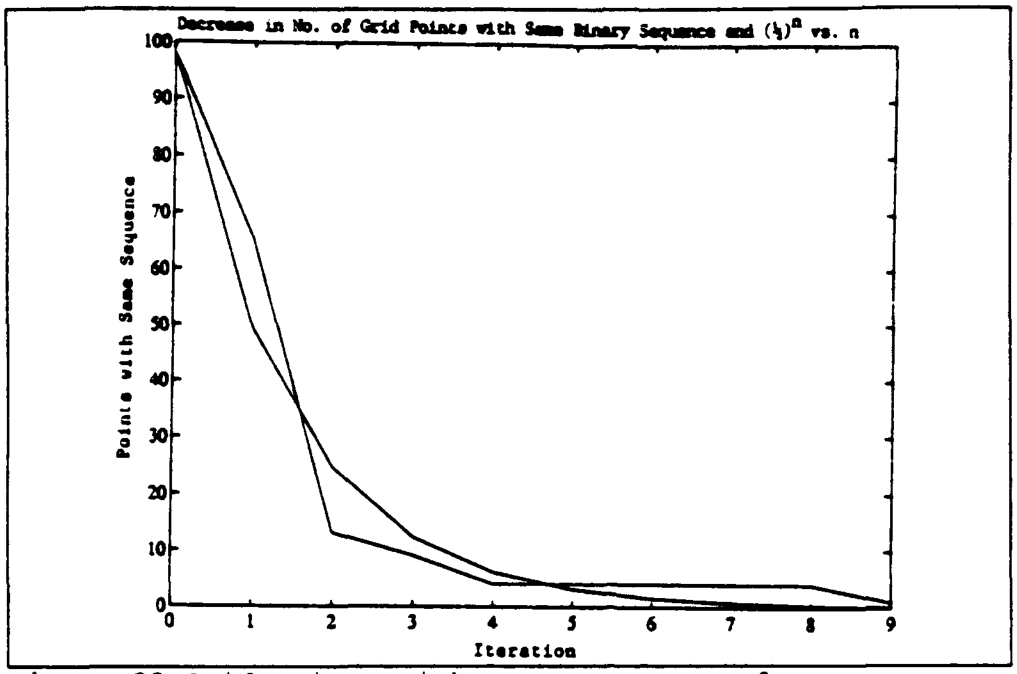

Figure 22 Grid points with same sequence after n iterations

Presumably based on the map's sensitive dependence on initial conditions there occurs a natural spreading effect, most noticeable in the left half of the quadrilateral, that forces the previously *close* points away from each other. A typical example of a run of this procedure using the left quadrilateral can be found in Appendix B.

Does there always exist an iterate N where the sequence for (W,Z) diverges from all sequences corresponding to even an extremely fine grid? Certainly this example only gives a taste of the capability of the computer to fill the right or left quadrilateral with points. Based on the limitation of computer memory it is not possible to use a Cartesian spacing small enough to model even a small fraction of the huge number of computer representable points. This is not our goal. We concern ourselves, instead, with the trend in the iterate N where all the grid points diverge from the iterate of (W,Z).

Let N be the iterate at which all the sequences corresponding to the grid points diverge from the sequence corresponding to (W,Z). Using our last example we plot the grid point that lasted until iterate N (call it (u,v)) with (W,Z) for the 9 iterations. The trajectory of the two points is depicted in Figures 23 through 32. Point (W,Z), again, is depicted as the cross (+) and the point (u,v) by an (x).

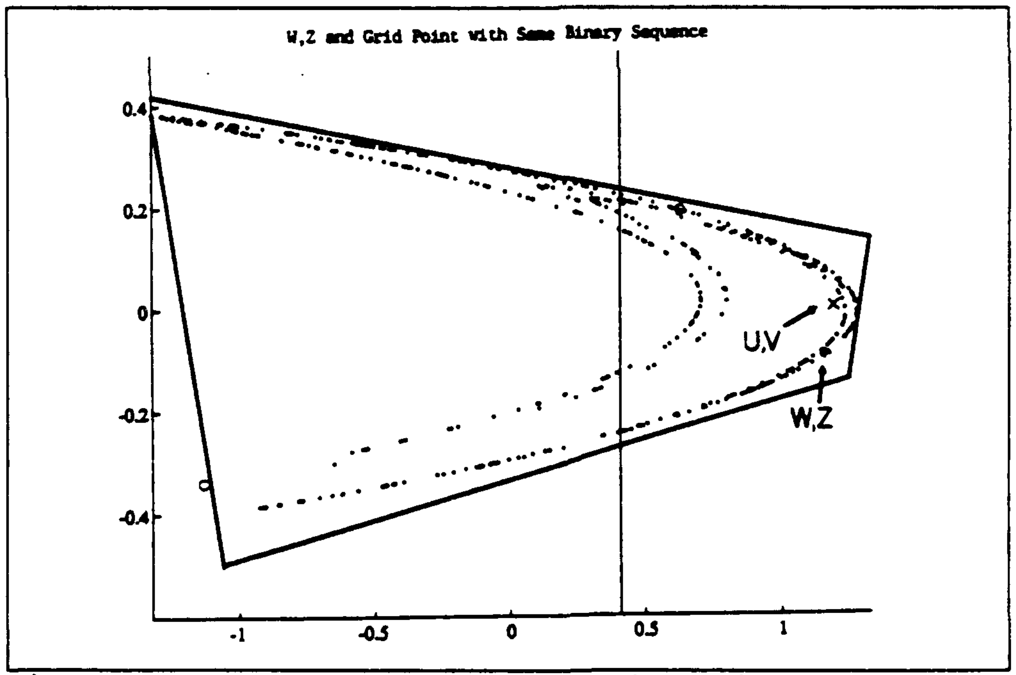

Figure 23 (W,Z) and (u,v) prior to iteration

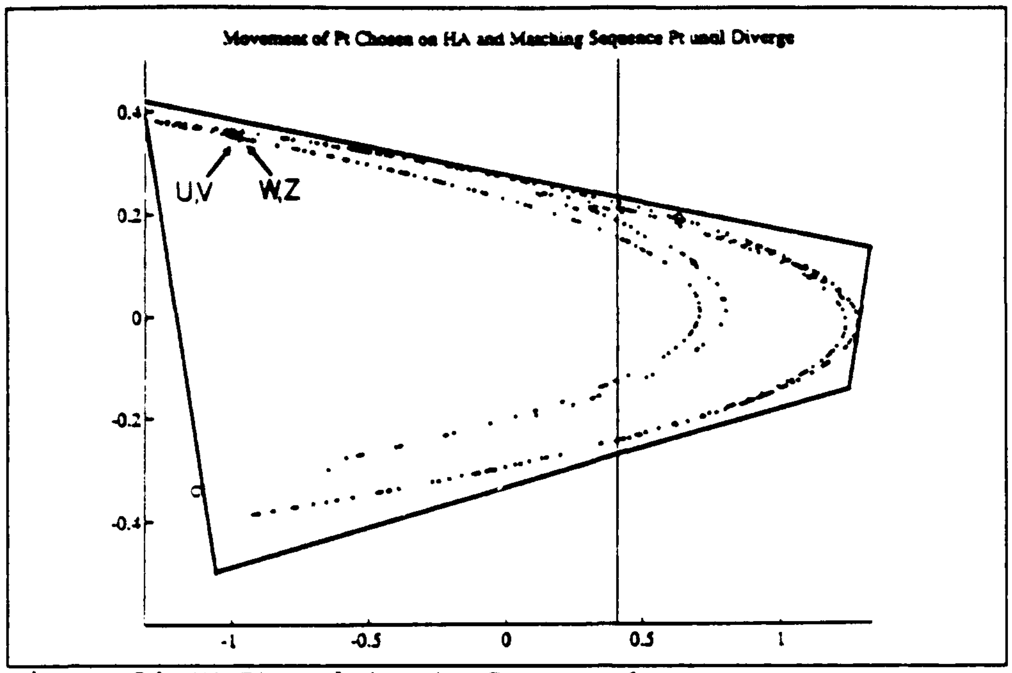

Figure 24 (W,Z) and (u,v): Iterate 1

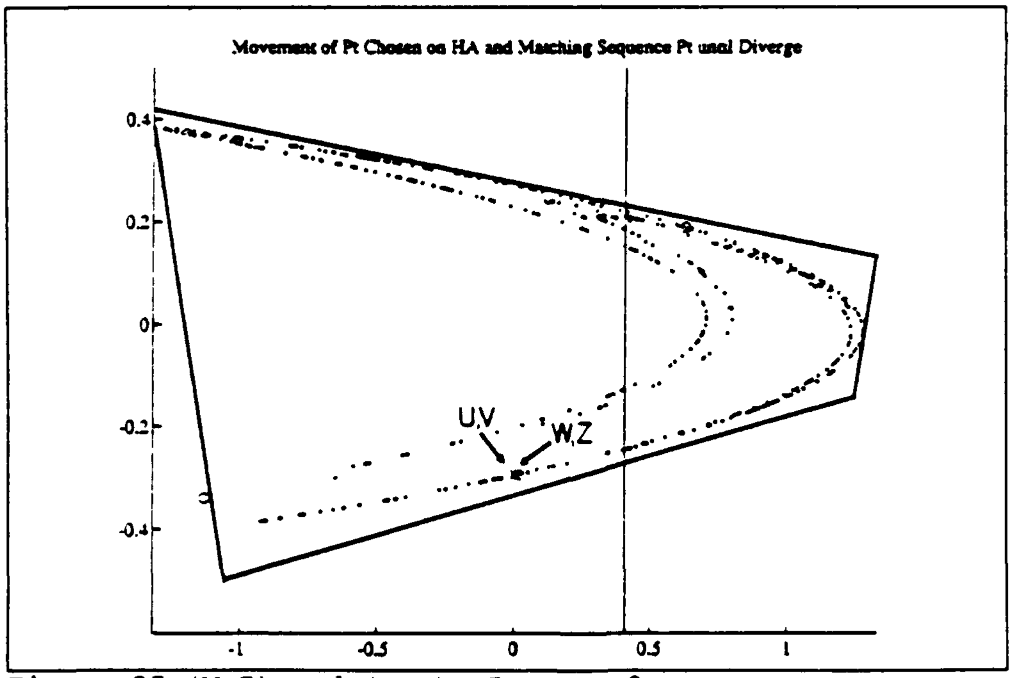

Figure 25 (W,Z) and (u,v): Iterate 2

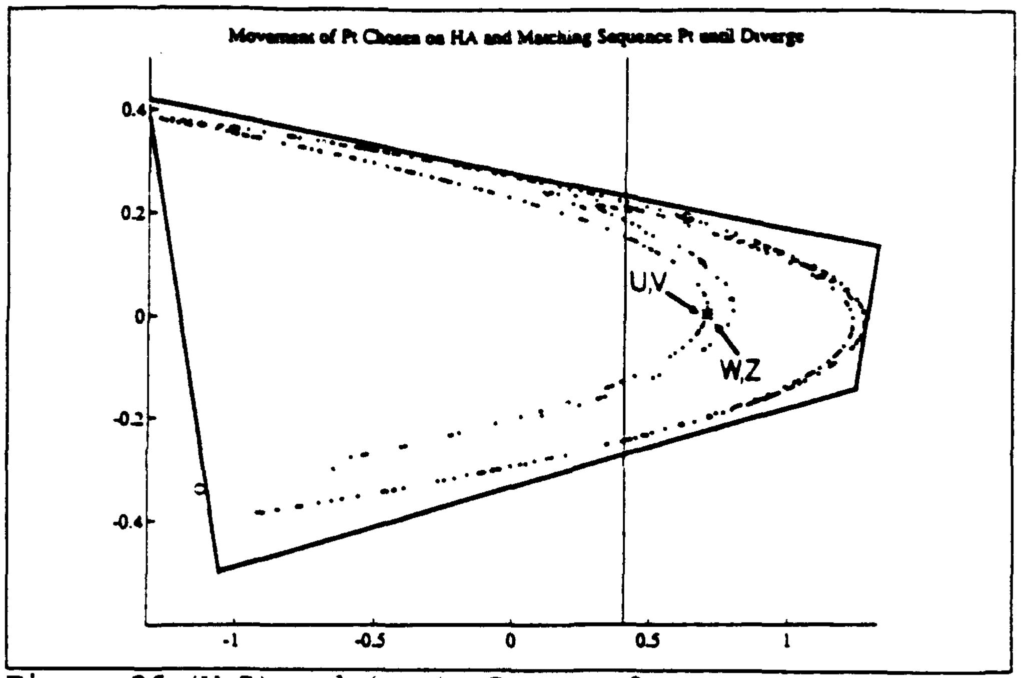

Figure 26 (W,Z) and (u,v): Iterate 3

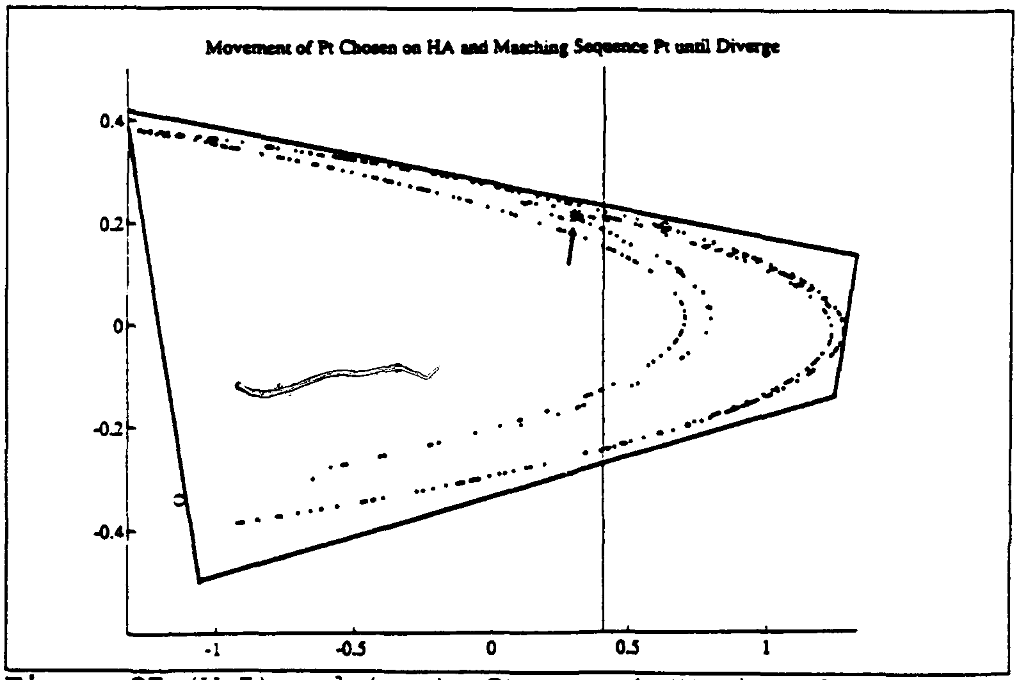

Figure 27 (W,Z) and (u,v): Iterate 4 (Notice that the two points are essentially superimposed).

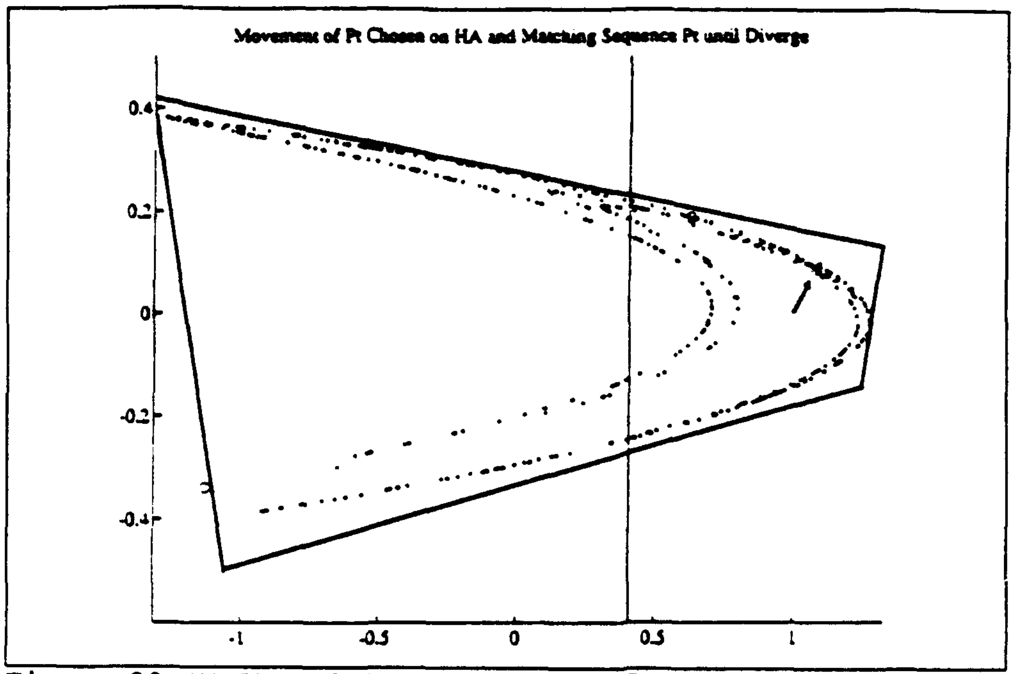

Figure 28 (W,Z) and (u,v): Iterate 5 (Notice that the two points are essentially superimposed).

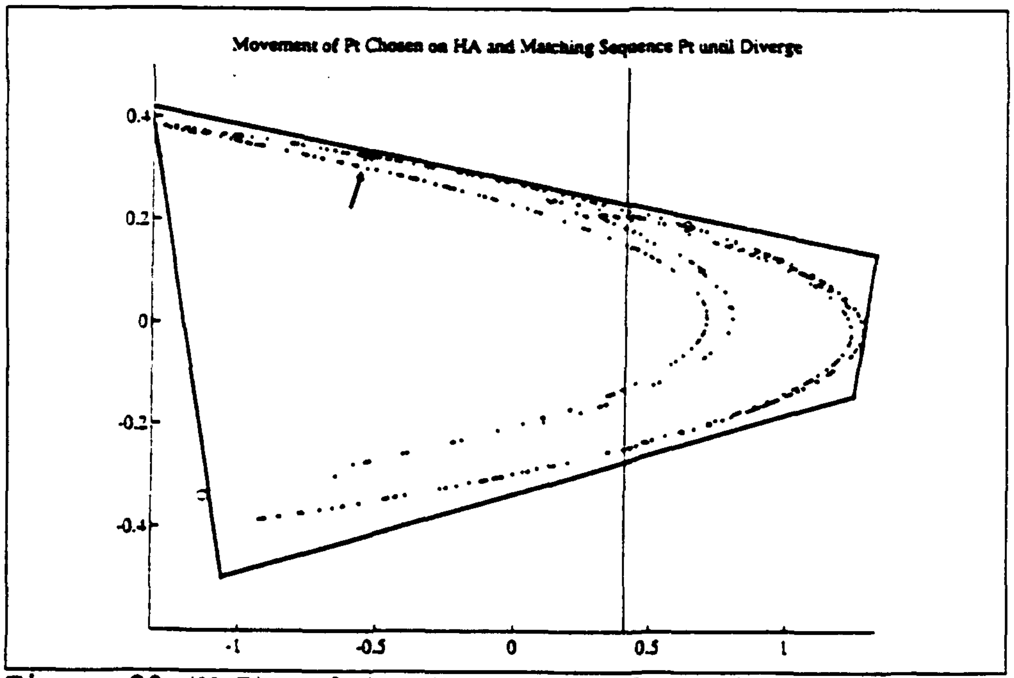

Figure 29 (W,Z) and (u,v): Iterate 6 (Notice that the two points are essentially superimposed).

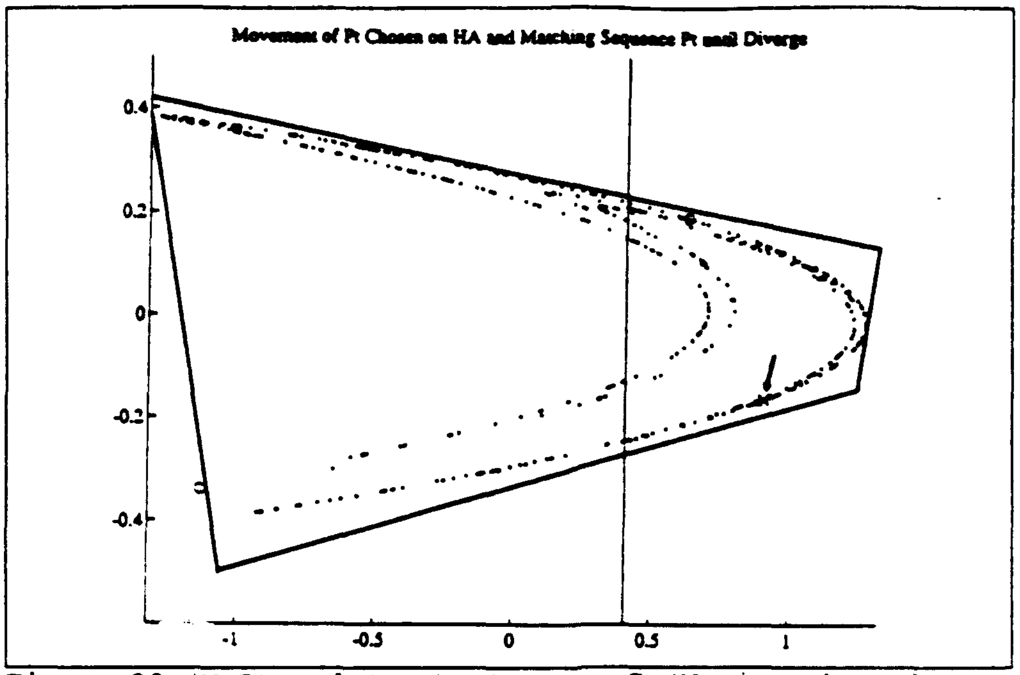

Figure 30 (W,Z) and (u,v): Iterate 7 (Notice that the two points are essentially superimposed).

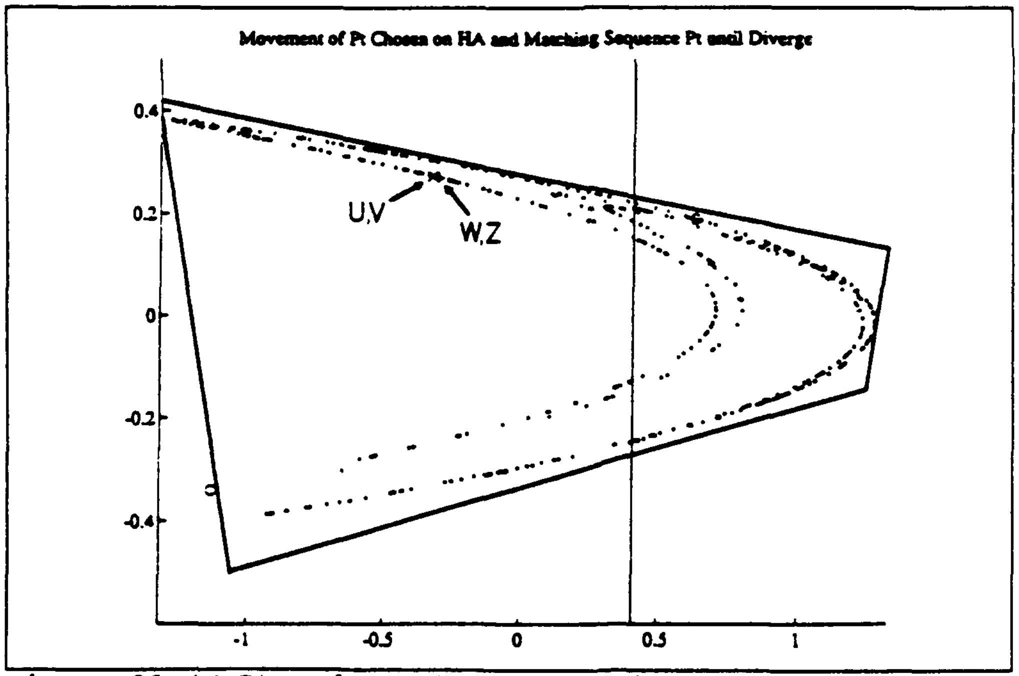

Figure 31 (W,Z) and (u,v): Iterate 8

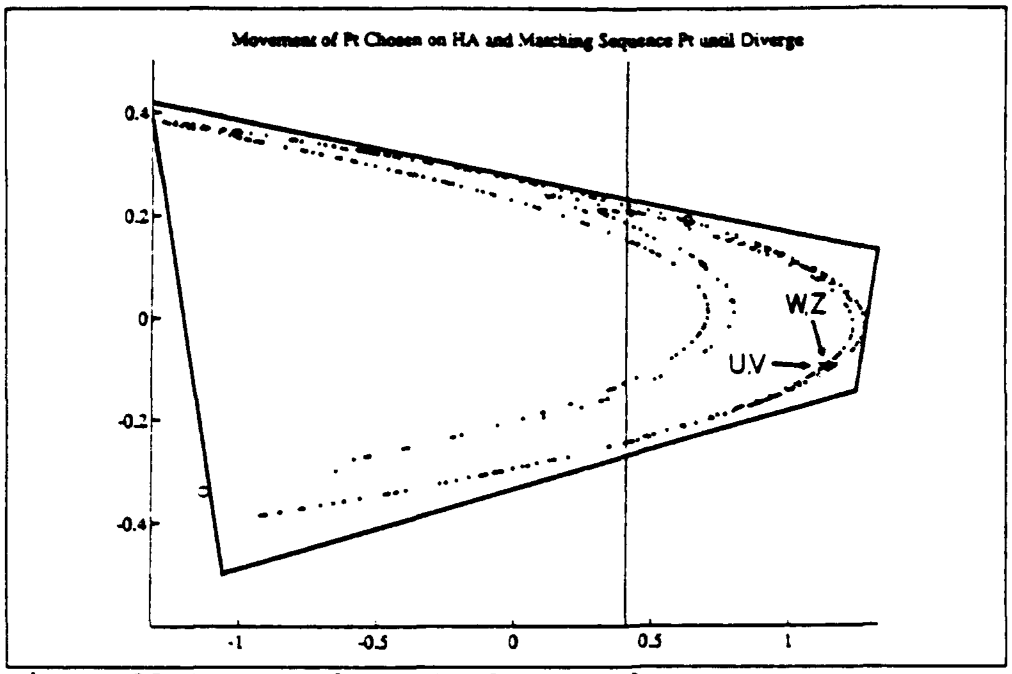

Figure 32 (W,Z) and (u,v): Iterate 9

Notice the *proximity* of (W,Z) and (u,v). They are in no way superimposed. Indeed, in a Euclidean sense the two points are not close at all. These positions may seem acceptable but even when the spacing between points is dropped to the limits of computer memory the proximity of (W,Z) to (u,v) does not appreciably change from this typical example. It is a fact that throughout the trajectories of the two points there may be iterations where the points are virtually superimposed as in this example, but as a rule (W,Z) and (u,v) are not initially *neighbors*. Furthermore, as in the typical run using the left quadrilateral in Appendix B, sometimes more than a single point (u,v) gives the same binary sequence as (W,Z). Typically one, but up to seven points (u,v), in arbitrary observations, have mirrored the bitstream of (W,Z) and these multiple matches likewise are not tightly packed points. This phenomenon lends support to our model because it is not the points closest to (W,Z) which typically give the same binary sequence. Typically many other points were much closer in a Euclidean sense to (W,Z) but they were rejected. That is, we cannot look only in the *immediate* vicinity of (W,Z) for points that offer good bitstream matches. Certainly, if a cryptographer knew that his chances for regenerating an identical binary sequence were much higher in the *vicinity* of (W,Z) he could use it to his advantage. Therefore, this quality is beneficial.

The GRDCOMP1.M - GRDCOMP4.M models (mentioned previously) perform a *grid comparison* between the chosen point (W,Z) and the grid developed on a particular side of the dynamic median. The models GRDCOMP1.M and GRDCOMP3.M examine behavior in the right quadrilateral, and GRDCOMP2.M and GRDCOMP4.M model the left quadrilateral. The GRDCOMP1.M and GRDCOMP2.M models use a predetermined (W,Z) value from their respective side of the quadrilateral but not on the attractor (H_A) to compare against the developed grid. The GRDCOMP3.M and GRDCOMP4.M models differ from GRDCOMP1.M and GRDCOMP2.M in that (W,Z) is iterated through the Hénon map to ensure that the point neighbors the attractor before the comparisons begin.

These computer models give us a feel for the N value to expect when we designate a particular spacing in the right or left grid field. The computer models PROOF1.M - PROOF4.M are identical to the GRDCOMP1.M - GRDCOMP4.M models except that they allow us to vary the spacing. We are able to designate a *coarse* and *fine* spacing range and an increment to use between these values. As the Cartesian spacing is steadily decreased and more points fill the particular *half-field*, by observation, N likewise seems to increase steadily. This behavior is pictured in Figure 33. Here, we use the PROOF1.M model incrementing the spacing from .003 to .5 by .001 for a total of nearly 500 entries.

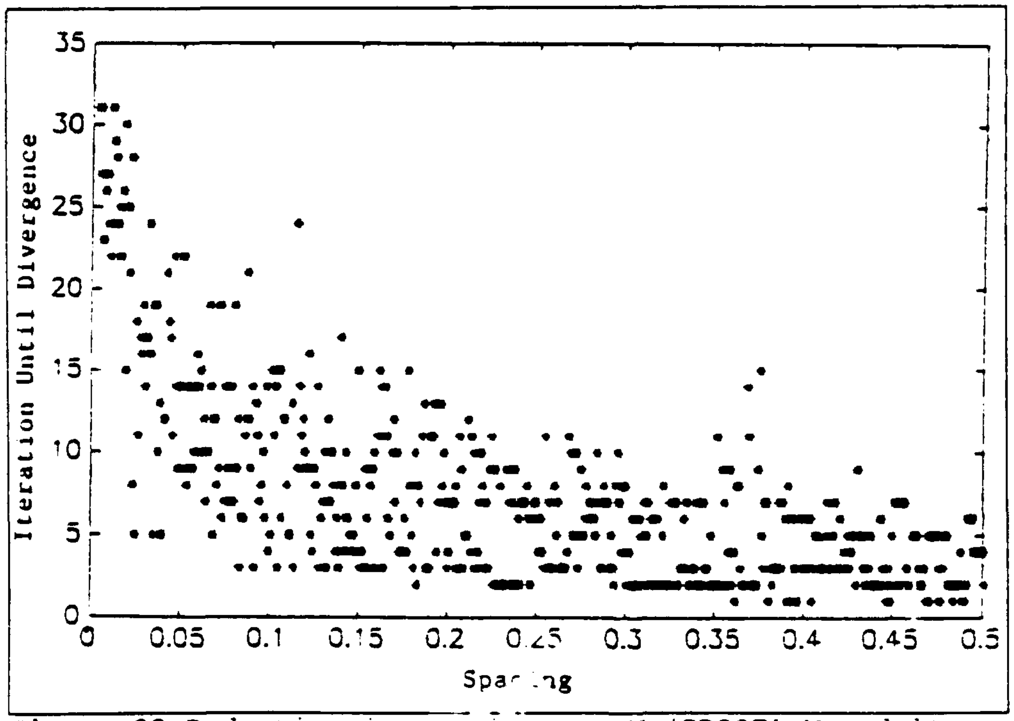

Figure 33 Reduction in spacing vs. N (PROOF1.M model)

Although there are a few outliers all the N values appear to be clustered and to steadily increase overall as the spacing becomes finer and finer. This behavior was observed for the PROOF2.M - PROOF4.M models as well.

Let us establish .10 as the *coarse spacing upper bound* because it is a round number and it allows us to fill either half-field with more than just a few points. In computer runs using this upper bound and steadily finer spacing (to the limits of computer memory) we observe the same general relationship between the decreasing spacing and N values. That is, as the spacing is steadily decreased the value of N correspondingly increases.

To further test our models certainly we must not limit ourselves to a comparison of a single (W,Z) value over the range of spacing values. Because we observe the same general behavior in our GRDCOMP models whether we use a particular (W,Z) not on the attractor or a (W,Z) from the attractor let us choose an arbitrary number of (W,Z) values and repeat our models (PRF1.M - PRF4.M). For simplicity the arbitrary number of (W,Z) values (31) are taken from a line belonging to the particular side of the quadrilateral. Figure 34 shows a run of model PRF1.M (which corresponds to our GRDCOMP1.M model but for multiple spacings and (W,Z) values).

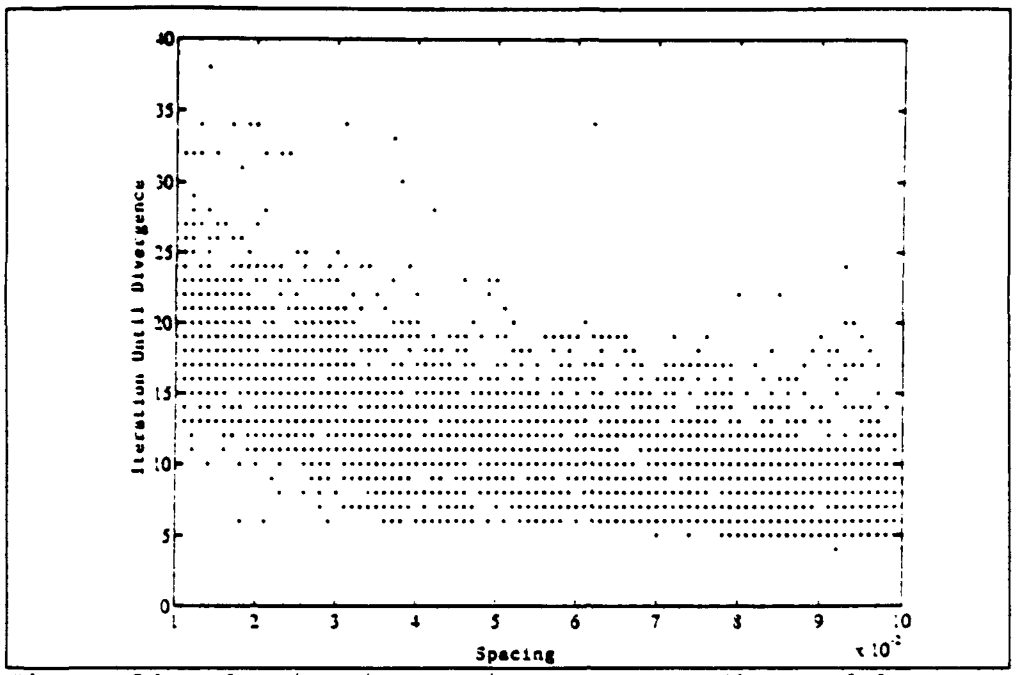

Figure 34 Reduction in spacing vs. N (PRF1.M model)

This run again is typical of all runs of the models PRF1.M - PRF4.M. This run increments the spacing from .01 to .10 in steps of .001. Notice that there do not appear to be 31 values for each spacing value. This phenomenon actually supports our observations because many of the points are superimposed on each other on the graph. Again, we observe an apparently steady increase in N values as the spacing becomes finer with few significant outliers. This graph contains almost 3000 points.

In order to show experimentally that the observed behavior exists, we select a modest fine spacing as a lower bound and a tiny increment. By using an extremely small spacing we are able to collect an enormous amount of data to use as statistical evidence. The limitation is not computer memory but the mainframe graphical statistical system (AGSS) which allowed a sample size of 45136 points. For each of the 31 (W,Z) values 1456 separate runs were completed. This corresponded to an increment of .0000625 between .01 and .10. The results are pictured in Figures 35-38.

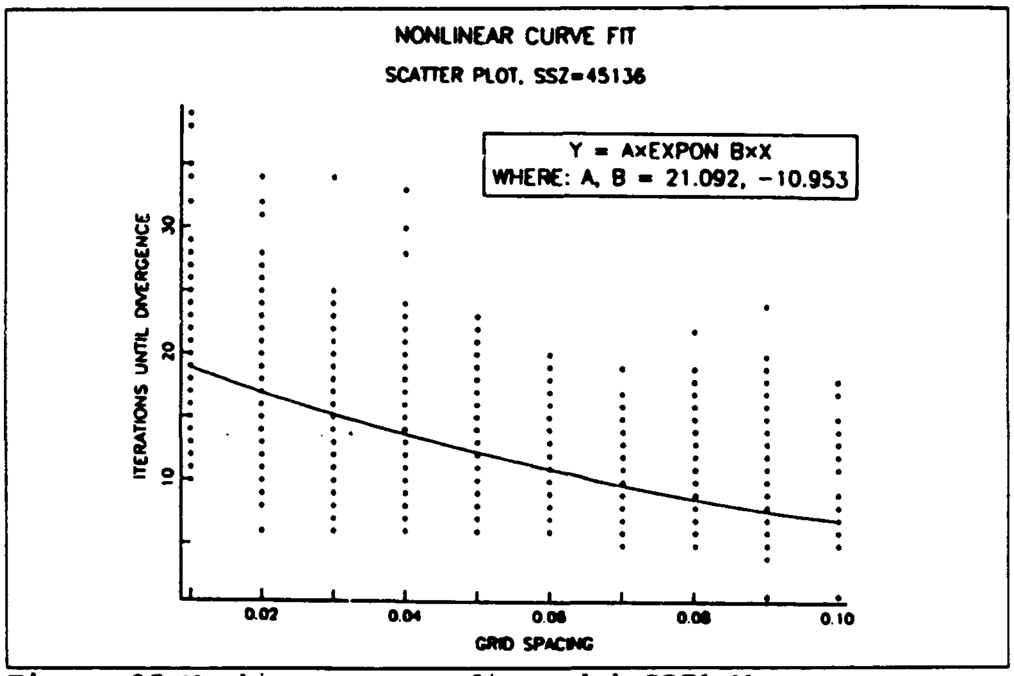

Figure 35 Nonlinear curve fit model PRF1.M

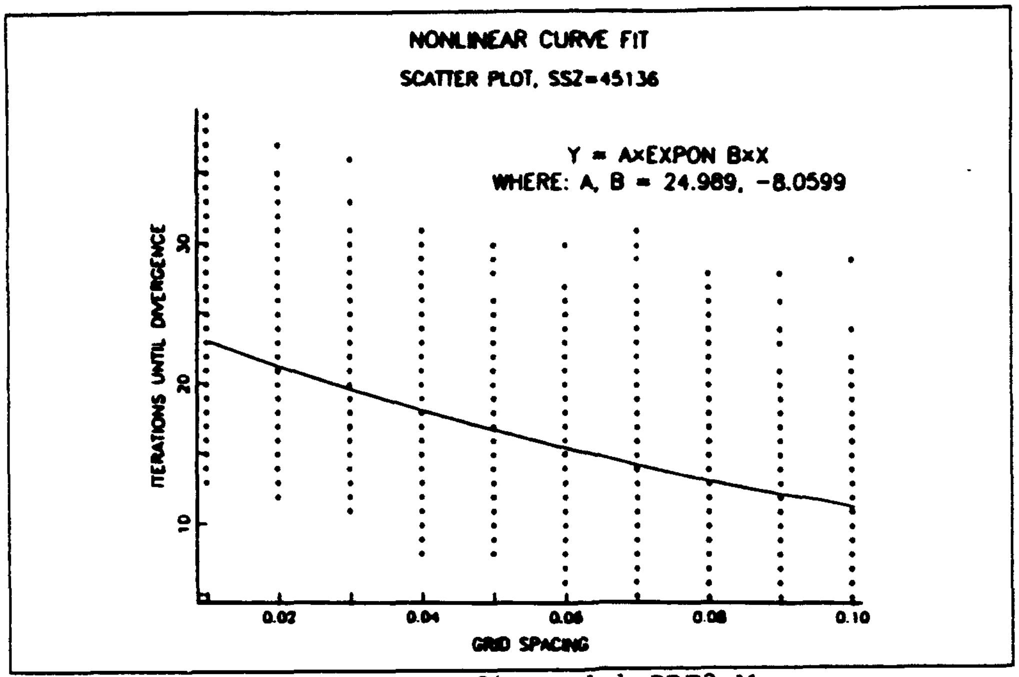

Figure 36 Nonlinear curve fit model PRF2.M

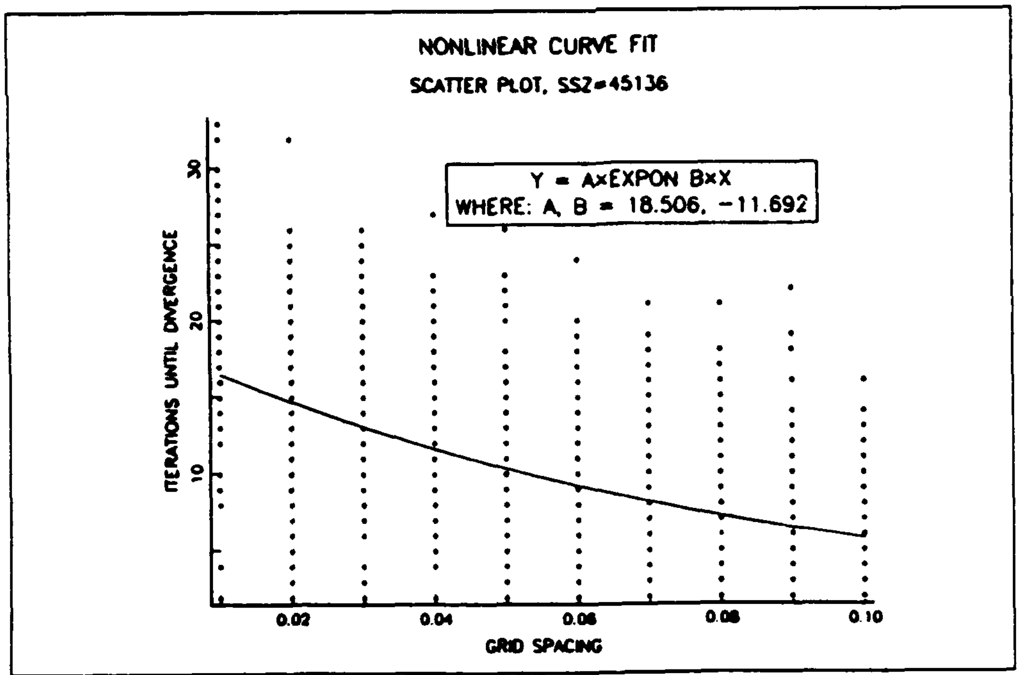

Figure 37 Nonlinear curve fit model PRF3.M

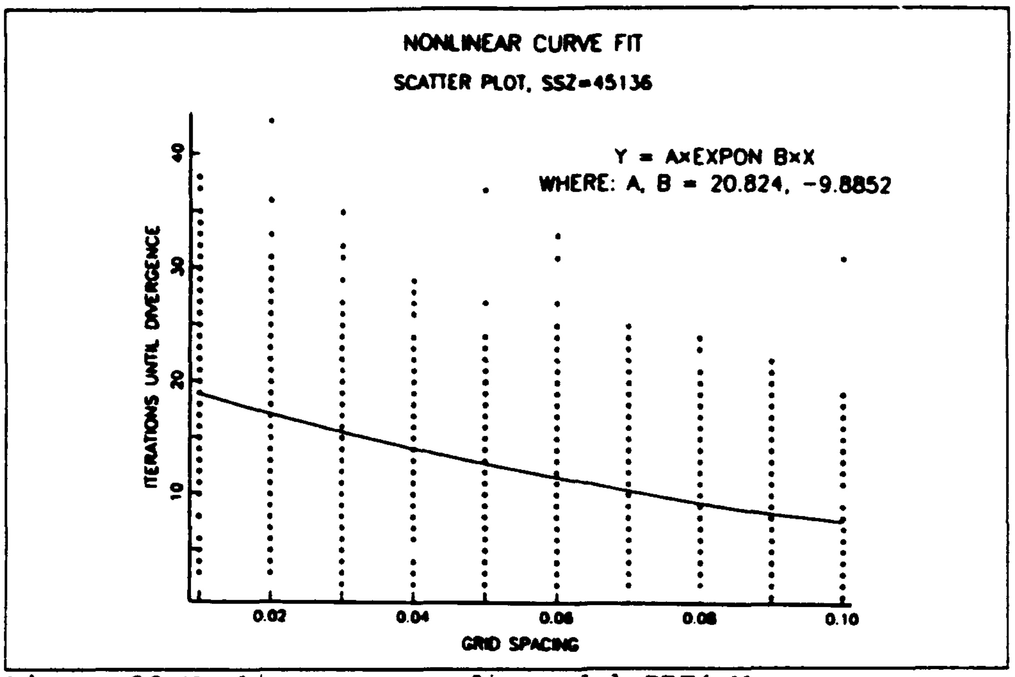

Figure 38 Nonlinear curve fit model PRF4.M

The figures correspond to the models PRF1.M to PRF4.M. The nonlinear curve fit takes into account all 45136 points although only a representative handful are pictured. The curves respectively correspond to the following nonlinear equations:

PRF1.M:	curve 1	$21.092\ e^{-10.933x}$
PRF2.M:	curve 2	$24.989\ e^{-8.0599x}$
PRF3.M:	curve 3	$18.506\ e^{-11.692x}$
PRF4.M:	curve 4	$20.824\ e^{-9.8852x}$

All four nonlinear curve fits are monotonically increasing from right to left. This result seems to suggest that in a *statistically acceptable fashion*, given a particular spacing for our grid we can predict an N value for which the binary sequence of (W,Z) will diverge from those of the grid field.

How small can we make o r grid spacing before our models fail? Certainly, for any particular computer there will exist a value, call it ε, such that computer numbers within that tolerance will be considered the same. That is, if x and y are computer representable numbers and $|x-y| < \varepsilon$, then $x = y$.

The value of ε varies. Let computer epsilon be defined as a value, ε, below which $1 + \varepsilon = 1$; 386 Matlab version 3.5M computes this value as approximately $2.24 \times 10^{-\text{[illegible]}}$. However, the value of ε (using the same definition) such that $0 + \varepsilon = 0$ is on the order of 10^{-308} for the same system. One can speculate that at some diminutive spacing S, our model will fail. That is, there will exist a spacing where the computer will not be able to build the grid. The computer model will calculate the first point x in the rectangle which encompasses the quadrilateral. But, because the spacing S is smaller than the ε tolerance that the computer needs to recognize the next point y, x and y are *seen* as the same point. Only the first grid point will appear in the grid. However, this is in keeping with our definition of one-to-one for our presumed homeomorphism because if two sequences are wholly the same then they come from the same initial condition. That is, the sequences are generated from two initial conditions (points) that the computer recognizes to be the same in finite precision although the initial conditions would differ in infinite precision. Thus, we have experimentally shown that for the symbolic dynamics if we have two unequal initial conditions then after some iteration N the corresponding sequences *should* diverge. Therefore, we have experimental evidence to support the statement that the symbolic dynamics is one-to-one.

IV. ONTO

A. ONTO PROPERTY

Whether or not the proposed symbolic dynamics has the onto property previously mentioned is directly brought into question by the *runs anomaly* which was closely examined in reference 13. This property, defined in Chapter II. section A., informally states that under the symbolic dynamics, every possible sequence of zeros and ones is possible or *realizable*. If a pseudorandom number generator has a favorable runs property, all possible *n-tuples* (sequences of zeros and ones of length n) must not only be possible but their occurrences must be balanced. The *unbalanced* count of particular n-tuples is precisely the runs anomaly pointed out by Forré as the fatal flaw of her pseudorandom number generator.

The runs property of a binary sequence is tested by *counting* the occurrences of the 2^n different possible n-tuples. For example, in Heyman's study, typical binary sequences of length 10^4 were used to test 2-tuples on a computer. There are 2^2=4 2-tuples: {0,0},{0,1},{1,0},{1,1}. The program ONTO.M (Appendix A) counts the four different 2-tuples in a binary sequence. Figure 39 shows a bar graph which gives the count of the four 2-tuples for a typical binary sequence of length 5 x 10^5.

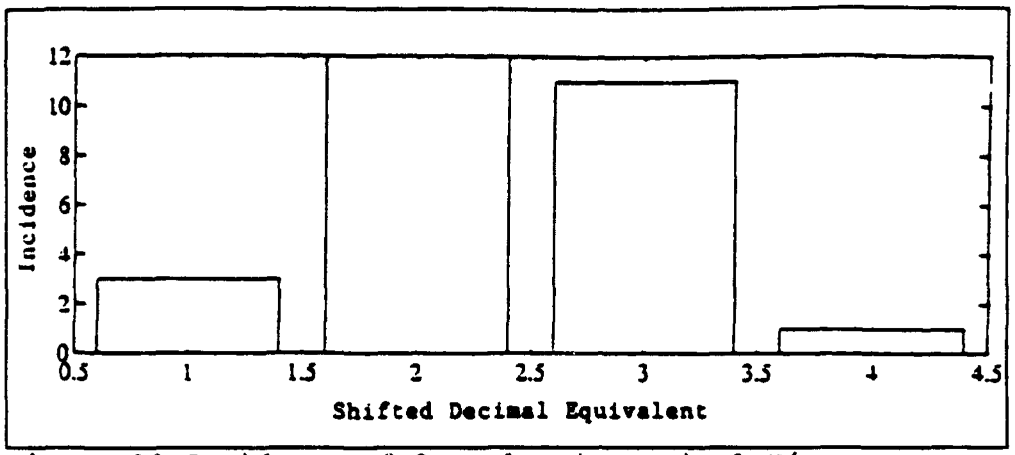

Figure 39 Incidence of 2-tuples in typical Hénon generated binary sequence

Notice that the horizontal axis is actually the decimal equivalent of the n-tuple shifted by one from zero. That is, instead of the following 2-tuple to decimal correspondence:

2-tuple		decimal	
00	-	0	where $[0\ 0] * [2\ 1]^T = 0$
01	-	1	where $[0\ 1] * [2\ 1]^T = 1$
10	-	2	where $[1\ 0] * [2\ 1]^T = 2$
11	-	3	where $[1\ 1] * [2\ 1]^T = 3$,

we use

00	-	1
01	-	2
10	-	3
11	-	4.

We will use this shifted correspondence for all n-tuples. The unbalanced or uneven *bins* of 2-tuples in Figure 39 demonstrates the runs anomaly. The depth of this anomaly is shown in Figures 40-42 which correspond to 3,4 and 5-tuple counts respectively for typical binary sequences of length 5 x 10^5. Modifications of ONTO.M were used to produce the associated data for finding the count of 2-tuples to 17-tuples.

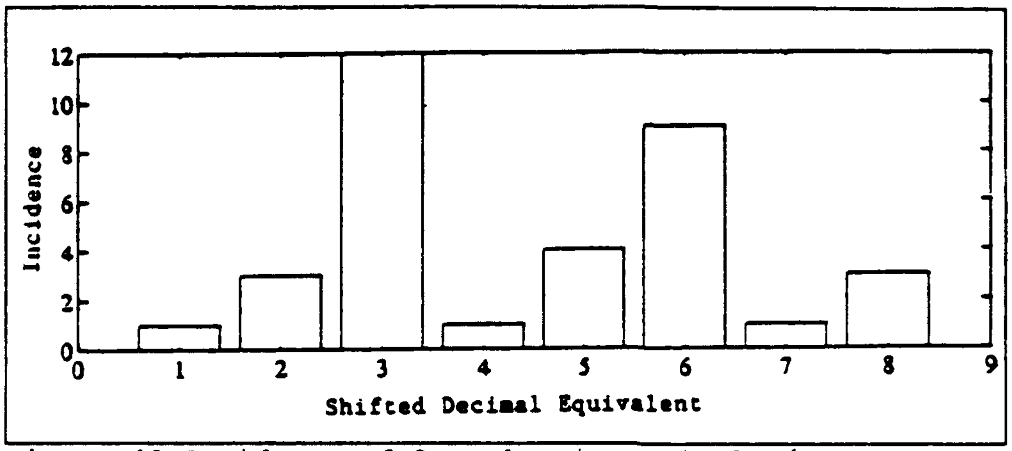

Figure 40 Incidence of 3-tuples in typical Hénon generated binary sequence

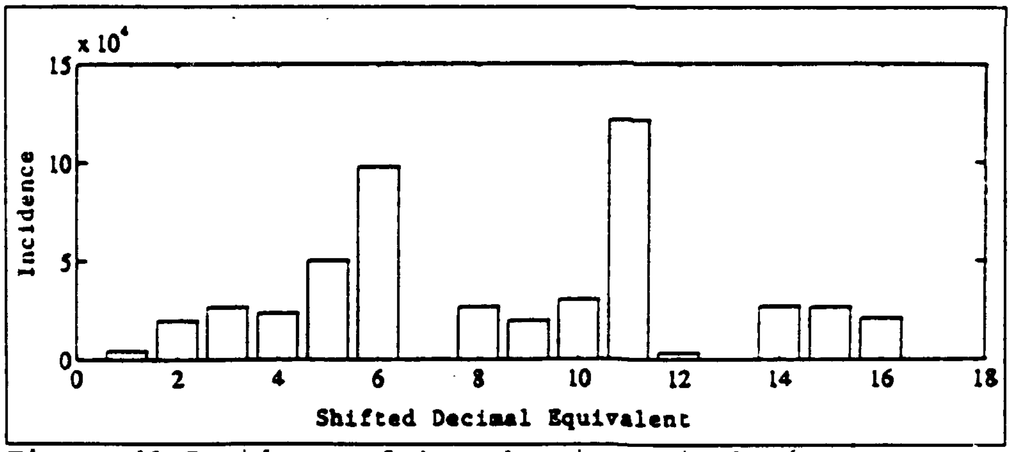

Figure 41 Incidence of 4-tuples in typical Hénon generated binary sequence

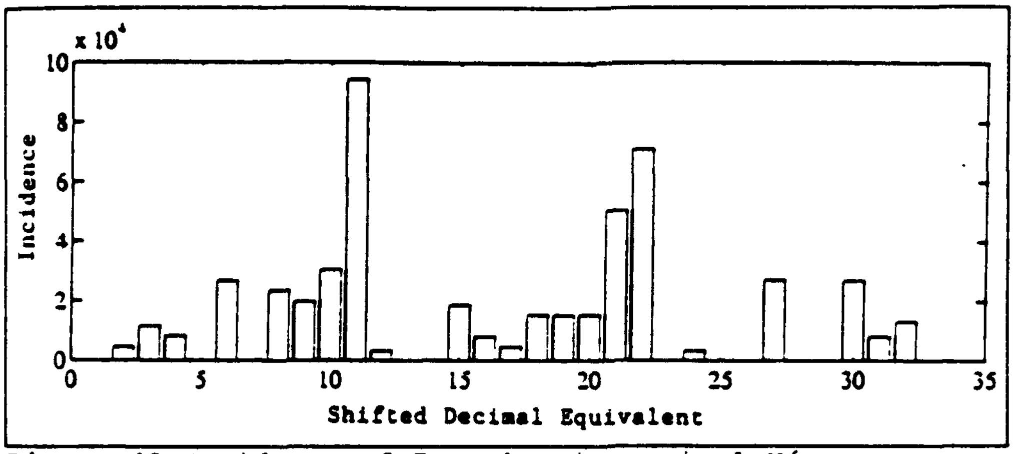

Figure 42 Incidence of 5-tuples in typical Hénon generated binary sequence

Notice that in Figure 41 the bin corresponding to the decimal 13 is empty. The bin corresponding to the decimal 7 is also empty but in tests using binary sequences of length 10^6 this bin is not. The 13 bin, however, remains empty in the 10^6 length test.

It was believed [Ref.13] that as sequences of greater and greater length were tested, the missing sequences would be found although the runs property, namely the balance of the bins, probably would not improve. Under memory constraints of the runs property test using a Sparc station 2, the length of the longest possible testable binary sequence was roughly 10^6. Using initial conditions from all four quadrants in separate tests with binary sequences of length 10^6 no 4-tuple was ever found to correspond to the decimal 13. The decimal 13 corresponds to the binary 4-tuple {1,1,0,0}. It is therefore expected that, when the 5-tuple runs are tested, those 5-

tuples which contain the 4-tuple {1,1,0,0} (see Table 2) should be empty.

TABLE 2 5-TUPLES CONTAINING {1,1,0,0} AND DECIMAL EQUIVALENTS

5-tuples with {1,1,0,0}	decimal equivalent
{1,1,0,0,0}	25
{1,1,0,0,1}	26
{0,1,1,0,0}	13
{1,1,1,0,0}	29

This is indeed the case; however, these are not the only subsequences of length 5 unrealized in a sequence of length 10'. Table 3 lists those additional 5-tuples that are not realized and their corresponding decimal values.

TABLE 3 5-TUPLES AND THEIR DECIMAL EQUIVALENT

5-tuple	decimal equivalent
{0,0,0,0,0}	1
{0,0,1,0,0}	5
{1,0,1,1,0}	23
{1,1,0,1,1}	28

In fact, the property was tested for n-tuples from n = 2 to 17 (see Appendix C for 6 through 16-tuple runs) and an increasing percentage of unrealized sequences occurs. Table 4 shows the number of unrealized sequences and the total number of possible n-tuples for a particular n.

TABLE 4 UNREALIZED SEQUENCES AND TOTAL POSSIBLE n-TUPLES

n-TUPLE	UNREALIZED SEQUENCES	POSSIBLE SEQUENCES	PERCENT UNREALIZED
2	0	4	0
3	0	8	0
4	1	16	6.25
5	8	32	25.00
6	28	64	43.75
7	75	128	58.59
8	179	256	69.92
9	399	512	77.93
10	856	1024	83.59
11	1794	2048	87.60
12	3715	4096	90.70
13	7628	8192	93.12
14	15580	16384	95.09
15	31588	32768	96.40
16	63993	65536	97.65
17	128922	131072	98.36

Figure 43 shows how the percentage of unrealizable sequences increases as the n-tuple length increases.

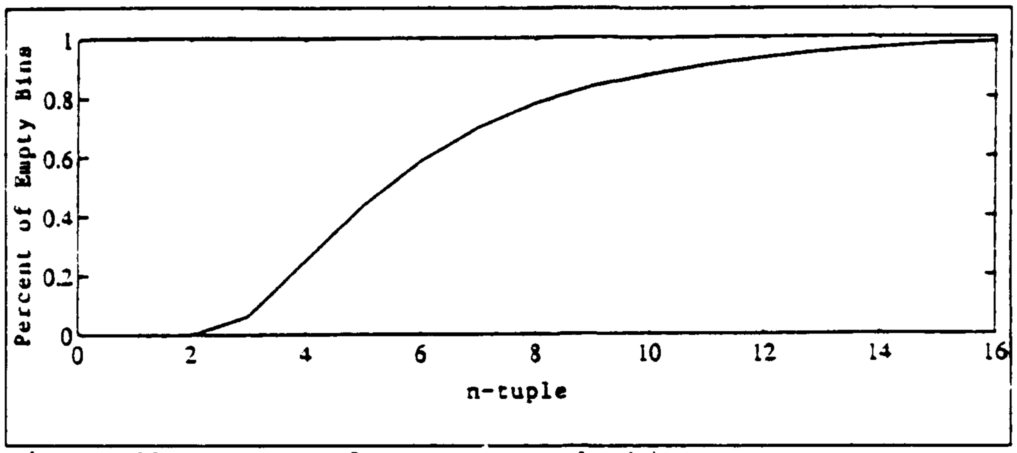

Figure 43 Percent of empty n-tuple bins vs. n

B. ANALYTIC PROOF OF UNREALIZABLE FOUR-TUPLE

Based on the previous data alone, the presumed homeomorphism cannot be disqualified from having the onto property. An analytically substantiated example of a wholly unrealizable sequence, however, would suffice. Let us investigate the first suspected unrealizable sequence, {1,1,0,0}. From the runs tests we see that the sequence {1,1,0,1} is possible although {1,1,0,0} is not. Figure 44 shows the portion out of 5000 points on the attractor which gives the sequence {1,1,0}.

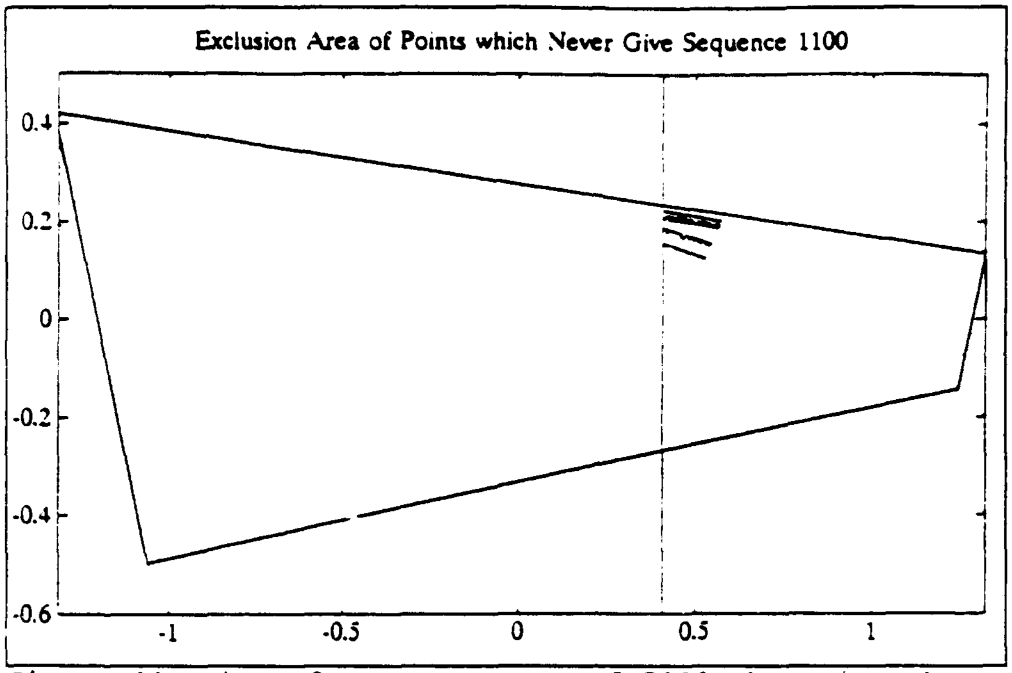

Figure 44 Points from a sequence of 5000 that give the sequence {1,1,0}

These are also the same points which give the sequence {1,1,0,1}, since none gives the sequence {1,1,0,0}. It is apparent that these points are localized in a particular area. Figures 45-47 show the subsequent iterates of these points.

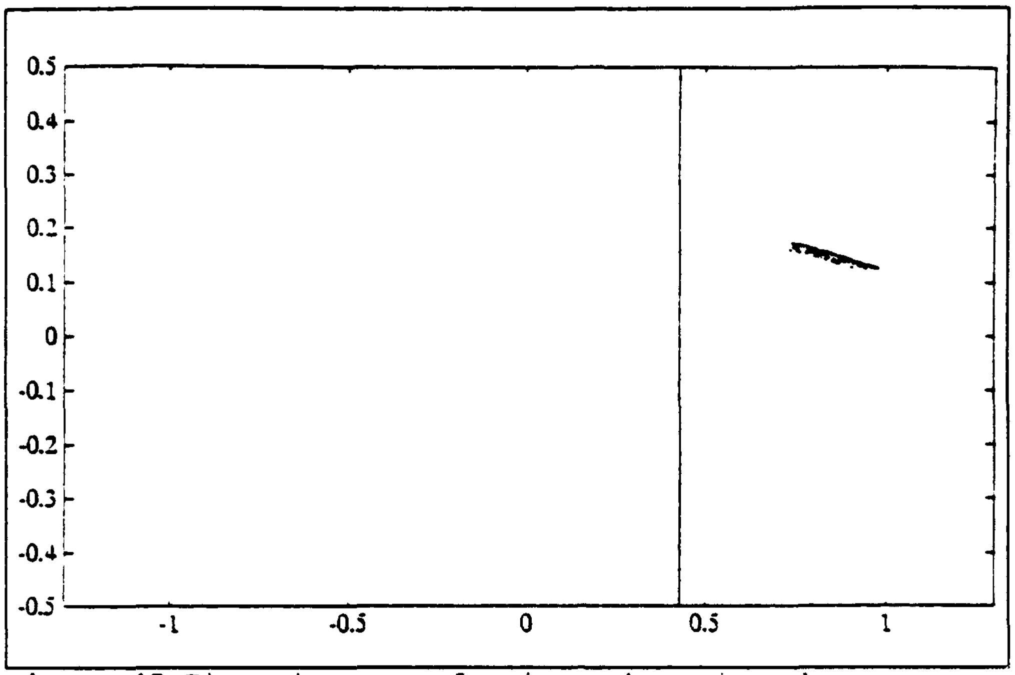

Figure 45 First iterate of points that give the sequence {1,1,0}

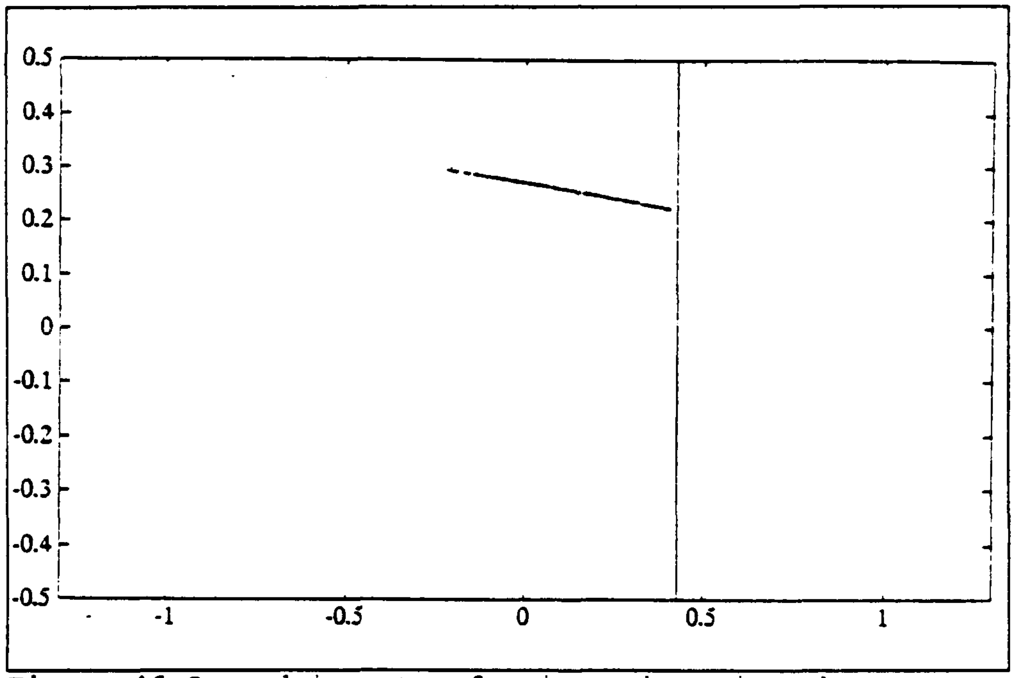

Figure 46 Second iterate of points that give the sequence {1,1,0}

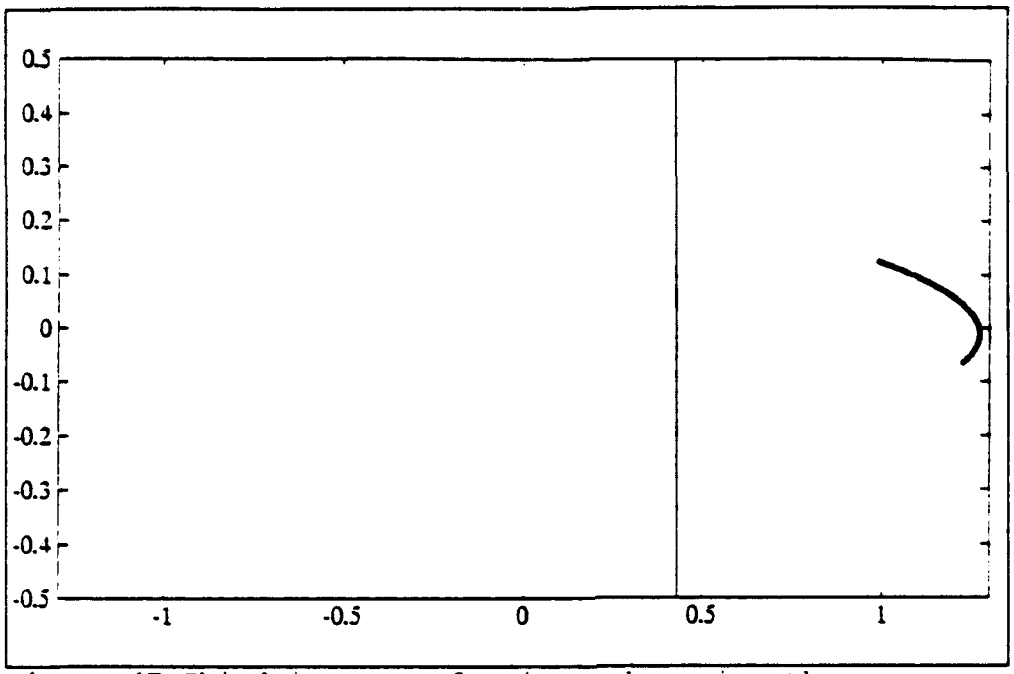

Figure 47 Third iterate of points that give the sequence {1,1,0}

It is clear that all these points under iteration correspond to the sequence {1,1,0,1}. A quadrilateral is placed around the points of Figure 44 in Figure 48 (call it the *subquadrilateral* or *subquad*).

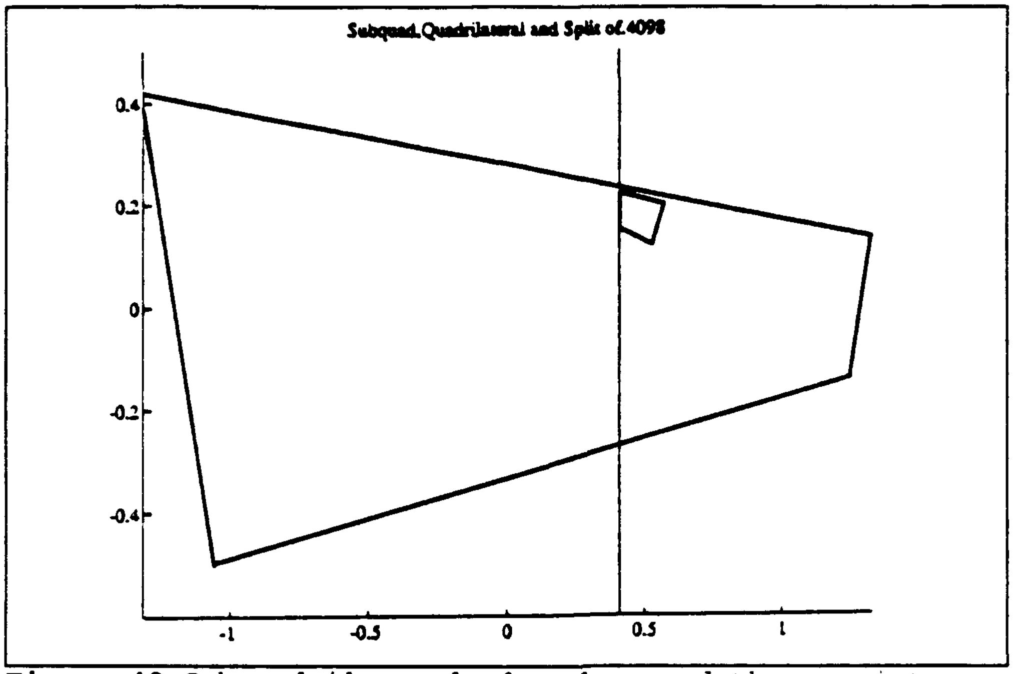

Figure 48 Subquadrilateral placed around those points that give the sequence {1,1,0}

By iterating the points belonging to this *subquad* using the program FIND110.M in Appendix A (see Figures 49-51) we see computer-generated *evidence* that no points within the subquad correspond to the binary sequence {1,1,0,0}.

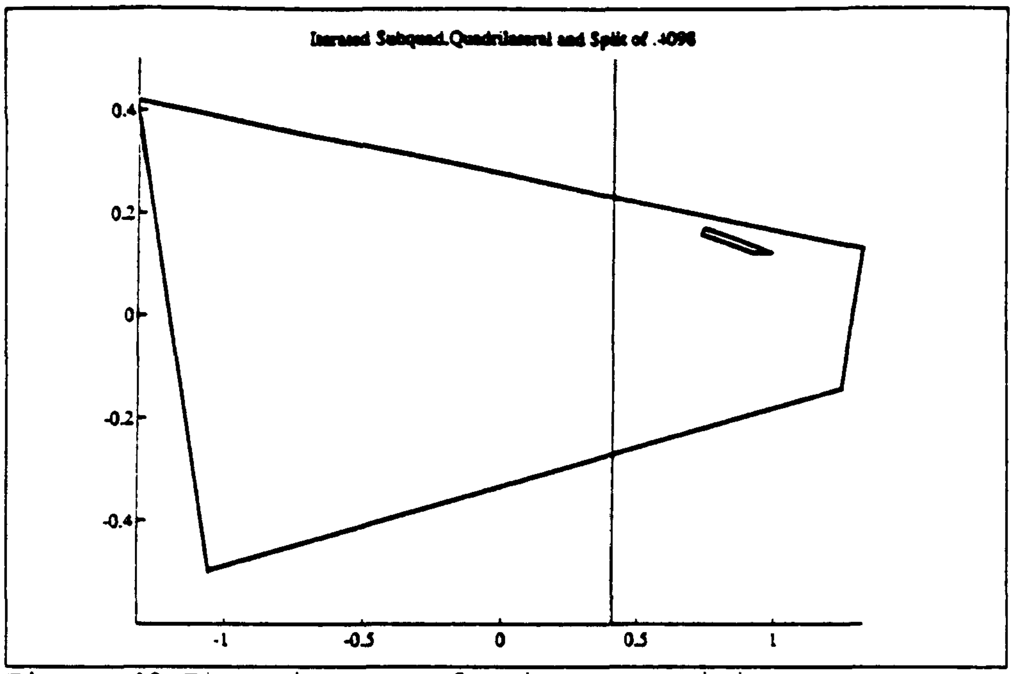

Figure 49 First iterate of points comprising subquadrilateral

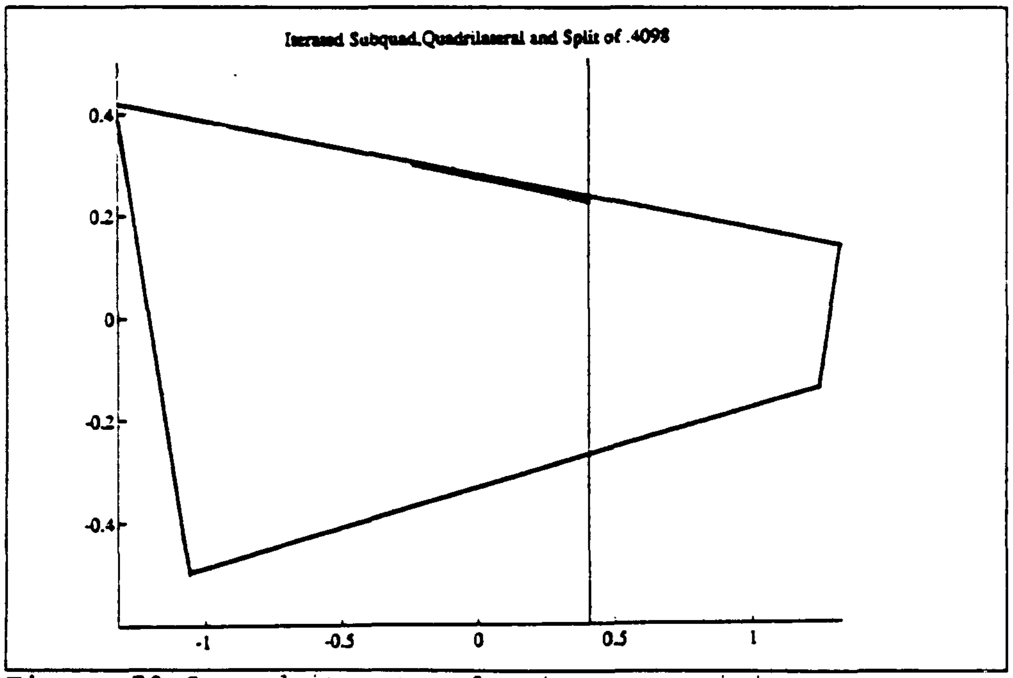

Figure 50 Second iterate of points comprising subquadrilateral

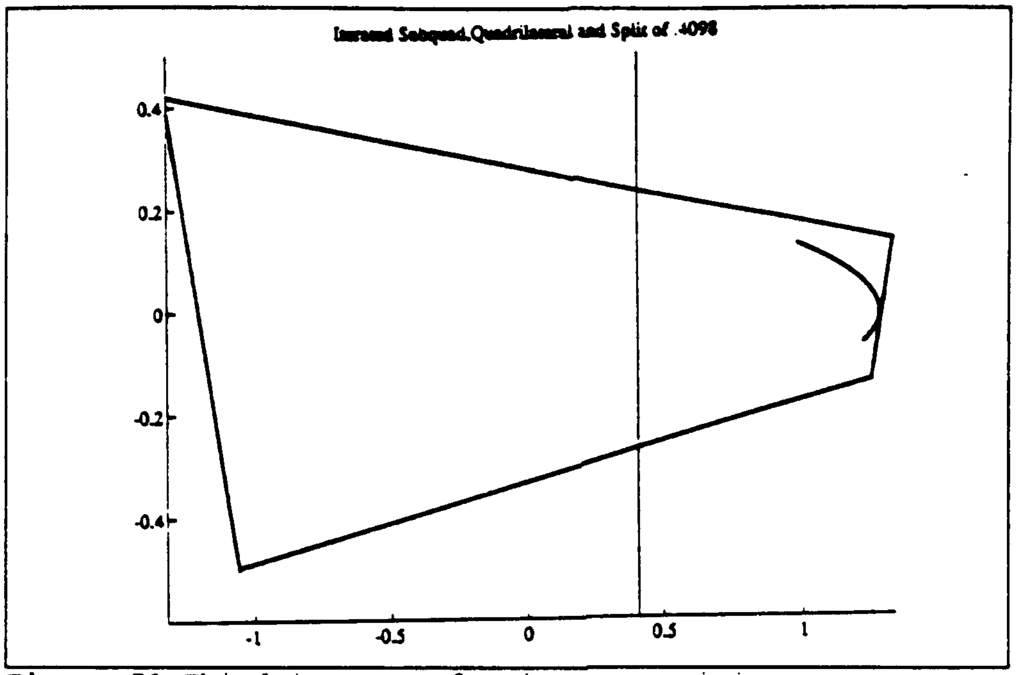

Figure 51 Third iterate of points comprising subquadrilateral

This *exclusion zone* does plainly *suggest* that the sequence {1,1,0,0} is unrealizable. Using a binary Hénon sequence of length 10^5, approximately 5115 points lie in the subquad or exclusion zone. This suggests that an orbit has a *probability density* of approximately 5.115% in the subquad. Since the Hénon map has been shown numerically to have topological transitivity, we expect that under reverse iteration of each point in the subquad there will be a preimage which will lie in the subquad. Therefore, this means that all points pass through the subquad and that no points which pass through this *window* give {1,1,0,0}.

Let us look more closely now at the points that **would** give {1,1,0,0} instead of those that would not. If we can show that the set of points that can possibly give {1,1,0,0} is the empty set, then we will have proved that this proposed symbolic dynamics does not possess the onto property. Recall the Hénon map introduced in Chapter I:

$$x_{n+1} = 1 - 1.4x_n^2 + y_n$$
$$y_{n+1} = .3x_n.$$

It follows that:

$$x_{n+1} = 1 - 1.4x_n^2 + .3x_{n-1}. \qquad (4.1)$$

Our first goal is to find the *solution set* that corresponds to {1,1,0}. In order for a sequence of (x,y) values to correspond to {1,1,0} the x values $\{x_{n-1}, x_n, x_{n+1}\}$ must obey the following inequalities:

$$x_{n-1} > x_{MED} \qquad (4.2)$$
$$x_n > x_{MED} \qquad (4.3)$$
$$x_{n+1} \leq x_{MED} \qquad (4.4)$$

where x_{MED}=.4098, the dynamic median rounded to four significant digits (previously discussed). Inequality (4.2) can be substituted in equation (4.1) as follows:

$$x_{n+1} > 1.12294 - 1.4x_n^2 \qquad (4.5)$$

where the constant 1.12294 is rounded to five significant digits as an underestimate in order to make our proof valid. Moreover, in all such cases we round in the appropriate (i.e. conservative) direction. In Figure 52 we see via (4.5) that the solution set corresponds to the region above the inverted parabola, in the $x_n x_{n+1}$-plane.

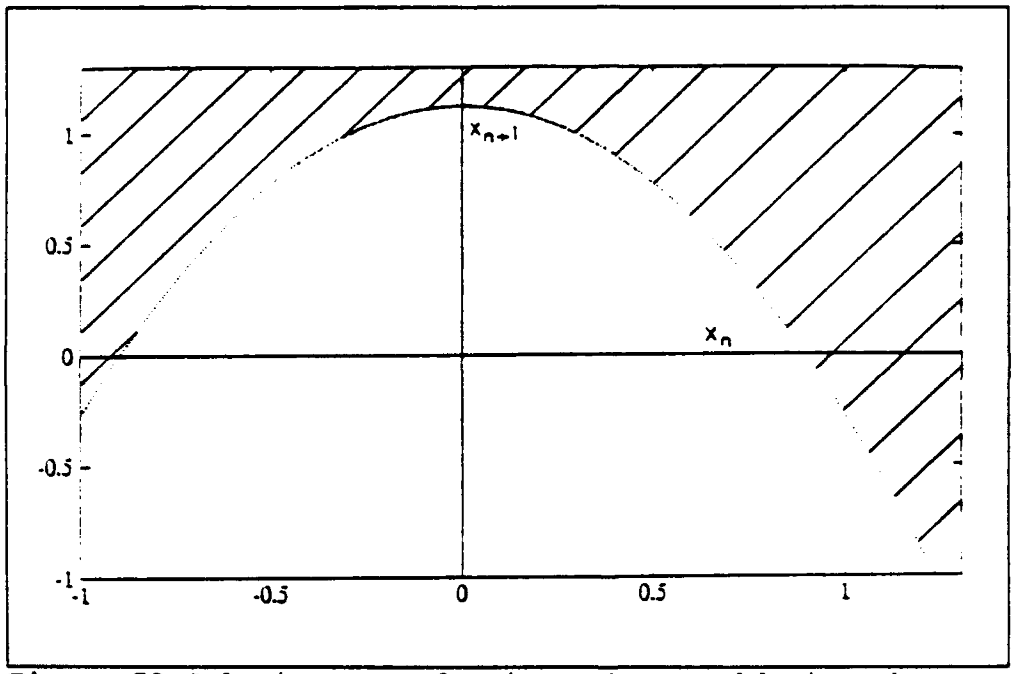

Figure 52 Solution set of points that could give the sequence {1,1,0} under x_{n-1} restriction

Figure 53 shows the possible solution set under restrictions (4.2),(4.3) and (4.4)

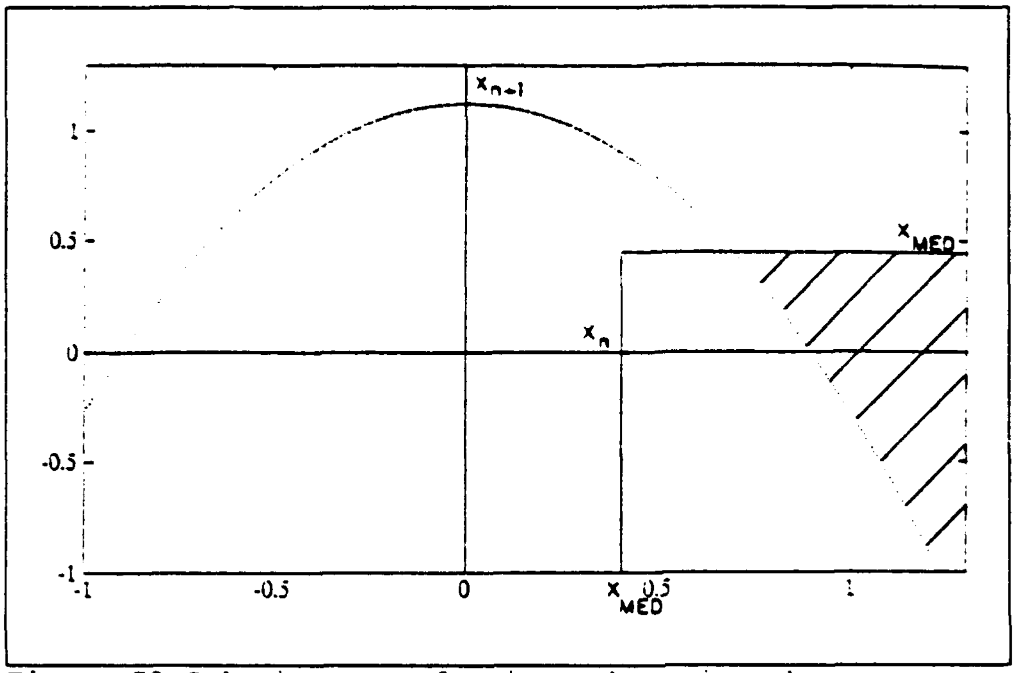

Figure 53 Solution set of points that give the sequence {1,1,0} under x_{MED} restrictions

By inequality (4.4), x_{n+1} has a maximum value less than or equal to x_{MED}. If we let x_{n+1} equal exactly x_{MED} in inequality (4.5) then x_n is further restricted to a value greater than .71371. Thus, the solution set corresponding to equation (4.1) is limited to the region shown in Figure 54 and reviewed as follows:

x_{n-1}	: region above the inverted parabola	: 1
x_n	: region such that $x_n > .71371$	: 1
x_{n+1}	: region such that $x_{n+1} \leq .4098$	: 0

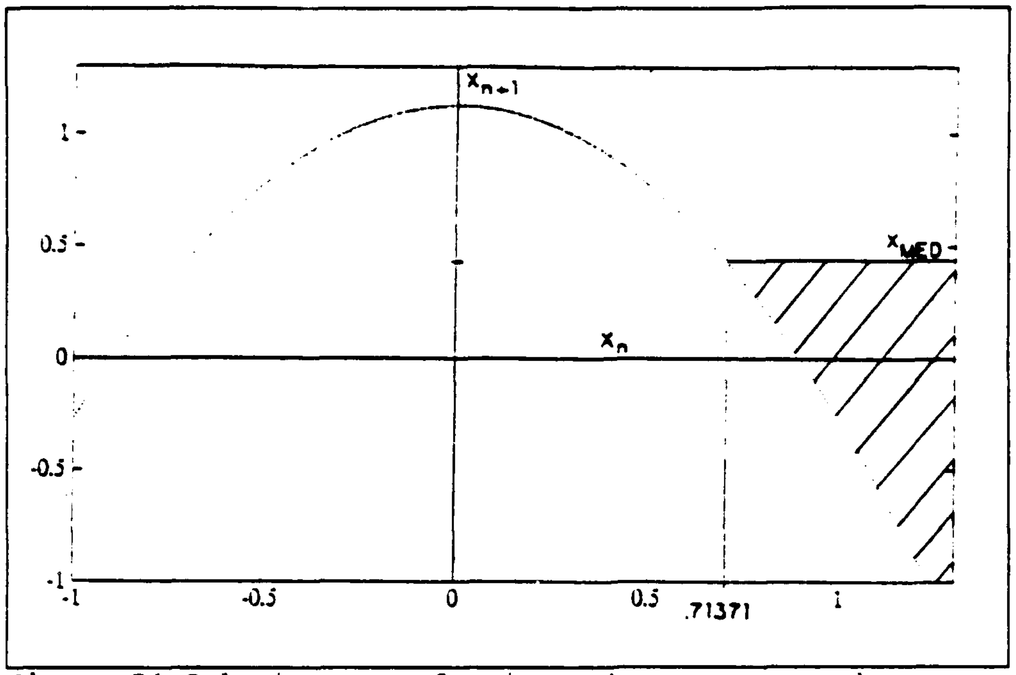

Figure 54 Solution set of points that generate the sequence {1,1,0}

By equation (4.1) it follows that if

$$x_{n+1} = 1 - 1.4x_n^2 + .3x_{n-1}$$

then

$$x_{n+2} = 1 - 1.4x_{n+1}^2 + .3x_n. \qquad (4.6)$$

To achieve the sequence {1,1,0,0}, x_{n+2} must be less than or equal to .4098. However, it is clear that the *largest* x_{n+2} value in equation (4.6) occurs at a minimum value for x_n and a maximum value for x_{n+1}, which lies in the solution region in Figure 54. The minimum x_n value and maximum x_{n+1} value are respectively .71371 and .4098. Using these optimum values in equation (4.6),

$$x_{n+2} \doteq 1 + .3(.71371) - 1.4(.4098)^2$$
$$\doteq .979003.$$

The symbol ($\doteq$) indicates that we round to the number of significant digits shown and that we round in the appropriate (conservative) direction as previously defined. This x_{n+2} value (not surprisingly) corresponds to the point with the minimum x value in Figure 51, that is, the third iterate of the subquad region which only gives the sequence {1,1,0,1}. Of course, this minimum x_{n+2} value exceeds .4098, and therefore, the sequence {1,1,0,0} is not realizable. Thus, we have found a counterexample to the proposed homeomorphism. Furthermore, it is shown in Appendix D that the sequence {0,0,0,0,0} is also analytically unrealizable.

V. CONCLUSIONS

A. RUNS ANOMALY-FATAL FACTOR OR LIMITING FACTOR?

We have seen that the classical Hénon map exhibits the attributes normally accepted as characterizing chaotic dynamical systems. These properties include sensitive dependence on initial conditions, topological transitivity and a dense set of periodic points. Despite this, however, we have provided ample numerical evidence and rigorous analytical proof that the proposed symbolic dynamics scheme $h:TR \rightarrow \Sigma_-$ for generating pseudorandom binary sequences is:

1) not a homeomorphism for the proposed symbolic dynamics scheme since it is not onto. (Because the Hénon attractor is a subset of TR we can also conclude that $h:H_A \rightarrow \Sigma_-$ is not onto.)

2) highly restricted in its viability as a pseudorandom number generator.

We have shown that not all binary sequences are generated with equal frequency. In particular we have shown that certain sequences are not realizable and that others are very sparsely attained. These facts support the observations and conclusions of Réjàne Forré which suggest that this scheme is unsuitable as a reliable means of generating pseudorandom numbers.

The evidence suggests that the *runs property* severely limits the potential of the scheme for widespread practical use. It is shown in reference 18 that subsequent elements of the binary sequence may actually be predicted with 70-80% accuracy by an artificial neural network (compared to 50% accuracy for a coin flip), most likely due to the severity of the runs anomaly. However, because of its simplicity and nonlinearity there may exist some applications for which the scheme would be well-suited. It is our belief that the general idea of using a chaotic discrete dynamical system to generate pseudorandom binary sequences, however, has merit and deserves additional study.

B. FUTURE WORK FOR AN IMPROVED SYMBOLIC DYNAMICS

Our numerical results reflect the basic structure of the attractor. Despite the lack of a homeomorphism the structural nature of the attractor is apparent in the binary output of the symbolic dynamics scheme. Although the attractor is accepted as possessing chaotic attributes, we believe that due to the structure of this "chaotic driver" (the classical Hénon map) pseudorandomness is not fully realized in this scheme. We believe that a sequence of more than four zeros is not possible because the left fixed point is both repelling and not on the attractor. Contrarily, it has been shown that a sequence of up to 23 ones is possible [Ref.16]. We conjecture (as suggested by computational evidence) that the sequences of ones are possible because the right quadrilateral fixed point is a saddle point. Specifically, the sequence of ones is possible due to the attracting axis of the saddle point. *This suggests that the symbolic dynamics scheme could be effective if there were such a saddle point on the attractor on both sides of the dynamic median x_{MED}.*

Since the location and nature of the fixed points for the Hénon map depend on the (a,b) parameters of the map, [Ref.1:p.170] there may exist parameter sets which provide this structure. The chaotic bands present in the bifurcation diagrams suggest there exist many other (a,b) pairs that may give even more complicated dynamics. Proving the existence of (a,b) parameters that give us these desired characteristics is only the first step. It is also required that a bounded attractor exist for the associated parameters. Furthermore, Forré had the luxury of being provided a numerically calculated trapping region from which to take the pseudorandom number generator key. This region, if it exists at all, would have to be recalculated for the new (a,b) pair.

If it could be proved that there does not exist an (a,b) pair which gives rise to saddle points that possess the previously mentioned attributes, then it may be possible to use another map which corresponds to a different attractor with these qualities. However, the major asset of the Hénon recurrence is its utter simplicity which translates to the fast generation of pseudorandom sequences.

Basing the symbolic dynamics scheme on the x value of each iterate is simple but it is not clear that it is the most effective. If we are willing to accept a more complicated scheme we could base our *split* on a different linear median or a nonlinear median. The goal in choosing a given median would be to bring parity to the runs property for the system of dual saddle points while retaining the other properties previously mentioned. A more complicated scheme might generate binary sequences more slowly, but the loss of speed may be warranted to improve the runs property.

It may be argued that despite an *improved system* there will still exist binary sequences that are unrealizable. However, the severity of the associated runs anomaly could be diminished should the shortest of these unrealizable sequences be of sufficient length. That is, we have seen that an unrealizable 4-tuple was catastrophic in this case because the problem of unrealizability translated to every subsequent n-tuple. In fact, we must anticipate that as described in Chapter IV even more n-tuples will be unrealizable than expected. The degradation of the system thus is directly related to the length (L) of the first unrealizable sequence(s). Since a pseudorandom number generator is judged by more than just the runs property criterion (see conclusion Ref.13) a system which possesses a "large" L-value may still provide an acceptable pseudorandom number generator.

APPENDIX A: MAIN PROGRAMS

NOTE: GRDCOMP PROGRAMS REQUIRE THE HENREAL PROGRAM TO RUN

```
% function [xmat, ymat] = grdcomp1(sp,W,Z)
% This function takes a point(W,Z)from the RT quad,initiates
a quad grid
% field based on a certain spacing, then iterates those grid
field points
% that match the binary string of (W,Z)[use(.1,.1) for now.
As long as
% the strings match, points are iterated using the Henon
recurrence. Only that
% (those) points that completely match are finally plotted
according to the
% following scheme:
%    b.    original grid field points
%    r*    original (W,Z)
%    go    iterated field points with matching binary string
at nth iteration
%    w+    iterated (W,Z) at nth iteration
%    gx    original xvec,yvec point(s) which matches binary
% string of W,Z. We also show how W,Z and
xvec(indices),yvec(indices) "walk" to the Henon attractor at
each iteration (with the attractor on screen).
  o=linspace(-1.33,1.32,500);
  s=-.1083*o + .276;
  u=linspace(1.32,1.245,500);
  v=3.64*u -4.6718;
  g=linspace(-1.06,1.245,500);
  h=.1533*g -.3344;
  e=linspace(-1.06,-1.33,500);
  f=-3.407*e -4.1119;
split = .4098;
    a = [-.1083 -1;-3.64 1;-.1562 1];
    b = [-.2760;-4.6718;-.3344];
    c = [split:sp:1.32];
    d = [-.6:sp:.5];
[x, y] = henreal(750,-1.0,-.25);
xline  =  [split split];        yline  = [-1.32 1.32];
lenc   = length(c);             lend   = length(d);
xvec   = zeros(1,lenc*lend);    yvec   = zeros(1,lenc*lend);
    if min((a*[w;z]) > b) == 0 | w < split
        disp('initial value is not in the right quadr')
        return;
    end
k = 0;
for p = 1:lenc
```

```
    for r = 1:lend
        k=k+1;
        if max((a*[c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k) = d(r);
        else
           xvec(k) = -10; yvec(k) = -10;
        end
    end
end
xvec = reshape(xvec,lenc*lend,1);
yvec = reshape(yvec,lenc*lend,1);
   m = find(xvec ~= -10 | yvec ~= -10);
xvec = xvec(m);  yvec = yvec(m);
xx   = xvec;        yy = yvec;
ww   = w;
zz   = z;
axis([-1.32 1.32 -.6 .5]);
indices = 1:length(xx);          % [1 2 3 4 . . .]
      i = 1;
   qvec = length(xx);
      q = indices;
   lenq = length(xx)                          %
initialization of q
  plot(xvec,yvec,'b.',W,Z,'r*',xline,yline,'w-');hold on;
  plot(o,s,'w.',u,v,'w.',e,f,'w.',g,h,'w.'); hold on;
title('Grid field and W,Z');pause;clg;hold off;
while length(q) > 1;
    axis([-1.32 1.32 -.6 .5]);
    x0 = yy + ones(size(yy)) - 1.4*xx.^2;
    y0 = .3*xx;
    w0 = zz + 1 - 1.4*ww.^2;
    z0 = .3*ww;
        if w0 > split
           qtemp = find(x0 > split | x0 == split);
        else
           qtemp = find(x0 < split);
        end;
        if length(qtemp) == 0,break;
        else q = qtemp;
        end
      i = i + 1;              % counts iterates where at
least one point matches
   lenq = length(q)
   qvec = [qvec,length(q)];% plot of how # with same binary
sequence
indices = indices(q);    % decreases each time through loop

   xx = x0(q);           % preserves to next iterate those
matching values of
   yy = y0(q);           % x0 and y0
   ww = w0;
   zz = z0;
pause;
   plot(xx,yy,'g.',w0,z0,'w+',xline,yline,'w-');hold on;
   plot(o,s,'w.',u,v,'w.',e,f,'w.',g,h,'w.');hold on;
title('Movement of points in grid with same binary
sequence')
   pause; clg;hold off;
end
axis;
vec = (1/2).^((1:length(qvec))-1);
plot((1:length(qvec))-1,qvec,'r',(1:length(qvec))-1,qvec(1)*
vec,'b');
title('Decrease in # of grid points with same binary
sequence vs 1/2^n')
pause;clg;
% In this way, we need the index key 'indices' to tell to
which grid point the
% surviving iterate corresponds with respect to the original
xvec  and yvec.
   axis([-1.32 1.32 -.6 .5]);
plot(xline,yline,'w',x,y,'b.',o,s,'w.',u,v,'w.',e,f,'w.',g,h
,'w.');hold on;
   plot(xvec(indices),yvec(indices),'gx',W,Z,'r*');hold on;
title('W,Z and the grid point with same binary
sequence');pause;
hold off;
xnew = xvec(indices);
ynew = yvec(indices); % now we show a plot of how W,Z and
the point
                 % that generates the same binary sequence
xnew,ynew
                 % walk around the attractor
axis([-1.32 1.32 -.6 .5]);
plot(xline,yline,'w',x,y,'b.',o,s,'w.',u,v,'w.',e,f,'w.',g,h
,'w.');
hold on;
xnewt  = xnew;
ynewt  = ynew;
    wt = w;
    zt = z;
for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wt.^2 + zt;
    z0t = .3*wt;
      wt = w0t;
      zt = z0t;
  xnewt  = x0t;
  ynewt  = y0t;
    plot(x0t,y0t,'gx',w0t,z0t,'r*');
```

```
title('Movement of pt chosen on HA and matching sequence pt
until diverge');
pause; plot(x0t,y0t,'ix',w0t,z0t,'i*');
end
hold off;
```

```
% function [xmat, ymat] = grdcomp2(sp,W,Z)
% This function takes a (W,Z) from the left quad,initiates a
quad grid field
% based on a certain spacing, then iterates those grid field
points that
% match the binary string of (W,Z) [use (-1,-.25) for now].
As long as the
% strings match, points are iterated using the Henon
recurrence. Only that
% (those) points that completely match are finally plotted
according to the
% following scheme:
%     b.    original grid field points
%     r*    original (W,Z)
%     go    iterated field points with matching binary string
at nth iteration
%     w+    iterated (W,Z) at nth iteration
%     gx    original xvec,yvec point(s) which matches binary
string of W,Z
% We then plot the iterates of W,Z and the point that most
closely matches
% its binary sequence.
o=linspace(-1.33,1.32,500);
s=-.1083*o+.276;
u=linspace(1.32,1.245,500);
v=3.64*u-4.6718;
g=linspace(-1.06,1.245,500);
h=.1533*g-.3344;
e=linspace(-1.06,-1.33,500);
f=-3.407*e-4.1119;
split = .4098;
    a = [3.4074 1; -.1083 -1; -.1562 1];
    b = [-4.1119; -.2760; -.3344];
    c = [-1.32:sp:split];
    d = [-.6:sp:.5];
[x, y] = henreal(750,-1.0,-.25);
  lenc = length(c);             lend = length(d);
 xline = [split split];        yline = [-1.32 1.32];
 xvec = zeros(1,lenc*lend);    yvec = zeros(1,lenc*lend);
    if min((a*[w;z]) > b) == 0 | w > split
        disp('initial value is not in left quadr')
        return;
    end
k=0;
for p =1:lenc
    for r = 1:lend
        k=k+1;
        if max((a*[c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k) = d(r);
        else
           xvec(k) = -10; yvec(k) = -10;
```

```
         end
      end
  end
  xvec = reshape(xvec,lenc*lend,1);     %makes a column vector
  yvec = reshape(yvec,lenc*lend,1);
     m = find(xvec~= -10 | yvec~= -10);
  xvec = xvec(m);    yvec = yvec(m);
  xx = xvec;         yy = yvec;
  ww = w;            zz = z;
     axis([-1.32 1.32 -.6 .5]);
  indices = 1:length(xx);                 % [1 2 3 4 . . .]
        i = 1;
     qvec = length(xx);
        q = indices;                                  %
  initialization of q
     lenq = length(xx)
     plot(xvec,yvec,'b.',W,Z,'r*',xline,yline,'w-'); hold on;
     plot(o,s,'w.',u,v,'w.',e,f,'w.',g,h,'w.');hold off;
  title('Grid field and W,Z');pause;clg;hold off;
  while length(q) > 1;
      axis([-1.32 1.32 -.6 .5]);
      x0 = yy + ones(size(yy)) - 1.4*xx.^2;
      y0 = .3*xx;
      w0 = zz + 1 - 1.4*ww.^2;
      z0 = .3*ww;
         if w0 > split
            qtemp = find(x0 > split | x0 == split);
         else
            qtemp = find(x0 < split);
         end;
         if length(qtemp) == 0,break;
         else q = qtemp;
         end
        i = i + 1;
     lenq = length(q)
     qvec = [qvec,length(q)];
  indices = indices(q);
       xx = x0(q);          % preserves to next iterate those
  matching values of
       yy = y0(q);          % x0 and y0
       ww = w0;
       zz = z0;
  pause;
     plot(xx,yy,'g.',w0,z0,'w+',xline,yline,'w-');hold on;
     plot(o,s,'w.',u,v,'w.',e,f,'w.',g,h,'w.'); hold off;
  title('Movement of points in grid with same binary
  sequence');
     pause;clg
  end
     axis;
  vec = (1/2).^((1:length(qvec))-1);
```

```
plot((1:length(qvec))-1,qvec,'r',(1:length(qvec))-1,qvec(1)*
vec,'b');
title('Decrease in # of grid pts with same binary sequence
as iter incr')
pause;clg
% In this way, we need the index key 'indices' to tell to
which grid point the
% surviving iterate corresponds with respect to the original
xvec  and yvec.
   axis([-1.32 1.32 -.6 .5]);
   plot(o,s,'w.',u,v,'w.',e,f,'w.',g,h,'w.');hold on;
plot(xvec(indices),yvec(indices),'gx',W,Z,'r*',x,y,'b.',xlin
e,yline,'w');
title('W,Z and the grid point with the same binary
sequence');pause;
hold off;
  xnew = xvec(indices);
  ynew = yvec(indices);
   axis([-1.32 1.32 -.6 .5]);
plot(xline,yline,'w',x,y,'b.',o,s,'w.',u,v,'w.',e,f,'w.',g,h
,'w.');
   hold on;
 xnewt = xnew;
 ynewt = ynew;
    wt = w;
    zt = z;
for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wt.^2 + zt;
    z0t = .3*wt;
     wt = w0t;
     zt = z0t;
  xnewt = x0t;
  ynewt = y0t;
   plot(x0t,y0t,'gx',w0t,z0t,'r*');hold on;
title('Movement of W,Z and matching sequence pt until
divergence');pause;
   plot(x0t,y0t,'ix',w0t,z0t,'i*');
end
hold off;
```

```
% function [xmat, ymat] = grdcomp3(sp,W,Z)
% This function takes a point(W,Z)from the RT quad and
iterates the point n
% times in order to ensure the point is on the Henon
attractor(ensuring
% that the nth iterate is in the left quadrant). It
initiates a grid field
% based on a certain spacing, then iterates those grid field
points that
% match the binary string of (wnew,znew) [use (-1,-.25) for
W,Z]. As long as
% the strings match, points are iterated using the Henon
recurrence. Only that
% (those) points that completely match are finally plotted
according to the
% following scheme:
%    b.    original grid field points
%    r*    original (W,Z)
%    go    iterated field points with matching binary string
at nth iteration
%    w+    iterated (W,Z) at nth iteration
%    gx    original xvec,yvec point(s) which matches binary
string of W,Z
% We also show how W,Z and xvec(indices),yvec(indices)
"walk" to the Henon
% attractor at each iteration (with the attractor on
screen).
o=linspace(-1.33,1.32,500);
s=-.1083*o + .276;
u=linspace(1.32,1.245,500);
v=3.64*u -4.6718;
g=linspace(-1.06,1.245,500);
h=.1533*g -.3344;
e=linspace(-1.06,-1.33,500);
f=-3.407*e -4.1119;
  split = .4098;
a = [-.1083 -1;-3.64 1;-.1562 1];
b = [-.2760;-4.6718;-.3344];
c = [split:sp:1.32];
d = [-.6:sp:.5];
[x, y] = henreal(750,-1.0,-.25);
 xline =  [split split];       yline = [-1.32 1.32];
 lenc  = length(c);            lend  = length(d);
 xvec  = zeros(1,lenc*lend);   yvec  = zeros(1,lenc*lend);
k = 0;
for p = 1:lenc
       for r = 1:lend
        k=k+1;
        if max((a*[c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k) = d(r);
        else
```

```
            xvec(k) = -10;  yvec(k) = -10;
         end
      end
   end
    xvec = reshape(xvec,lenc*lend,1);
    yvec = reshape(yvec,lenc*lend,1);
       m = find(xvec~= -10 | yvec~= -10);
    xvec = xvec(m);
    yvec = yvec(m);
   wattr = w;      % This preserves the values of W,Z
   zattr = z;
   for j = 1:40                                     % This
   ensures wattr,zattr is
       wattr(j+1) = 1.0 - 1.4*wattr(j)^2 + zattr(j);  % on the
   attractor
       zattr(j+1) = .3*wattr(j);
           if j > 20 & wattr(j+1) > split, break;
           end
   end
    wnew = wattr(j+1);       % preserves value on attractor
   wattr,zattr
    znew = zattr(j+1);
   wattr = wnew;
   zattr = znew;
      xx = xvec;          % preserves the values of xvec & yvec
      yy = yvec;
   axis([-1.32 1.32 -.6 .5]);
   indices = 1:length(xx);          % [1 2 3 4 . . .]
         i = 1;
      qvec = length(xx);
         q = indices;
      lenq = length(xx)                  % initialization of q
    plot(xvec,yvec,'b.',wattr,zattr,'r*',xline,yline,'w-');hold
   on;
     plot(o,s,'w.',u,v,'w.',e,f,'w.',g,h,'w.'); hold off;
   title('Grid field and point chosen on attractor');pause;clg;
   while length(q) > 1;
       axis([-1.32 1.32 -.6 .5]);
       x0 = yy + ones(size(yy)) - 1.4*xx.^2;
       y0 = .3*xx;
       w0 = znew + 1 - 1.4*wnew.^2;
       z0 = .3*wnew;
           if w0 > split
              qtemp = find(x0 > split | x0 == split);
           else
              qtemp = find(x0 < split);
           end;
           if length(qtemp) == 0,break;
           else q = qtemp;
           end
      i = i + 1;            % counts iterates where at least
one point matches
   qvec = [qvec,length(q)];% plot of how # with same binary
sequence
   lenq = length(q)
indices = indices(q);    % decreases each time through loop
     xx = x0(q);          % preserves to next iterate those
matching values of
     yy = y0(q);          % x0 and y0
   wnew = w0;
   znew = z0;
pause;
   plot(xx,yy,'g.',w0,z0,'w+',xline,yline,'w-');hold on;
   plot(o,s,'w.',u,v,'w.',e,f,'w.',g,h,'w.');
title('Movement of points in grid with same binary
sequence')
   pause; clg;hold off;
end
axis([1 2 3 4]);axis;
vec = (1/2).^((1:length(qvec))-1);
plot((1:length(qvec))-1,qvec,'r',(1:length(qvec))-1,qvec(1)*
vec,'b');
title('Decrease in # of grid points with same binary
sequence vs 1/2^n')
pause;clg;
% In this way, we need the index key 'indices' to tell to
which grid point the
% surviving iterate corresponds with respect to the original
xvec  and yvec.
   axis([-1.32 1.32 -.6 .5]);
plot(xline,yline,'w',x,y,'b.',o,s,'w.',u,v,'w.',e,f,'w.',g,h
,'w.');hold on;
plot(xvec(indices),yvec(indices),'gx',wattr,zattr,'r*');hold
on;
title('wattr,zattr & grid pt with same binary
sequence');pause;
hold off;
xnew = xvec(indices);
ynew = yvec(indices); % now we show a plot of how
wattr,zattr and the point
                % that generates the same binary sequence
xnew,ynew
                % walk around the attractor
axis([-1.32 1.32 -.6 .5]);
plot(xline,yline,'w',x,y,'b.',o,s,'w.',u,v,'w.',e,f,'w.',g,h
,'w.');
hold on;
xnewt  = xnew;
ynewt  = ynew;
wattrt = wattr;
zattrt = zattr;
```

```
for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wattrt.^2 + zattrt;
    z0t = .3*wattrt;
  wattrt = w0t;
  zattrt = z0t;
  xnewt  = x0t;
  ynewt  = y0t;
    plot(x0t,y0t,'gx',w0t,z0t,'r*');
title('Movement of pt chosen on HA and matching sequence pt
until diverge');
pause; plot(x0t,y0t,'ix',w0t,z0t,'i*');
end
hold off;
```

```
% function [xmat, ymat] = grdcomp4(sp,W,Z)
% ONLY DIFF BTWN 3&4 IS THAT 4 WEEDS QUADR,3 USES WHOLE
RECTANGLE.
% This function takes a point(W,Z) from the quadr and
iterates the point n
% times in order to ensure the point is on the Henon
attractor(ensuring
% that the nth iterate is in the left quadrant). It
initiates a grid field
% based on a certain spacing, then iterates those grid field
points that
% match the binary string of (wneW,Znew) [use (-1,-.25) for
W,Z]. As long as
% the strings match, points are iterated using the Henon
recurrence. Only that
%  those  points that completely match are finally plotted
according to the
% following scheme:
%    b.    original grid field points
%    r*    original (W,Z)
%    go    iterated field points with matching binary string
at nth iteration
%    w+    iterated (W,Z) at nth iteration
%    gx    original xvec,yvec point(s) which matches binary
string of W,Z
format long
% We also show how W,Z and xvec(indices),yvec(indices)
"walk" to the Henon
% attractor at each iteration (with the attractor on
screen).
o=linspace(-1.33,1.32,500);
s=-.1083*o + .276;
u=linspace(1.32,1.245,500);
v=3.64*u -4.6718;
g=linspace(-1.06,1.245,500);
h=.1533*g -.3344;
e=linspace(-1.06,-1.33,500);
f=-3.407*e - 4.1119;
  split = .4098;
a = [3.4074 1; -.1083 -1; -.1562 1];
b = [-4.1119; -.2760;  -.3344];
c = [-1.32:sp:split];
d = [-.6:sp:.5];
    if min((a*[w;z]) > b) == 0 | w > split
        disp('initial value is not in left quadr')
        return;
    end
[x, y] = henreal(750,-1.0,-.25);
 xline = [split split];     yline = [-1.32 1.32];
  lenc = length(c);          lend = length(d);
  xvec = zeros(1,lenc*lend); yvec = zeros(1,lenc*lend);
```

```
k=0;
for p = 1:lenc
    for r = 1:lend
        k = k + 1;
% This next section puts everything outside of the quad to
(-10,-10)
        if max((a * [c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k)=d(r);
        else
           xvec(k) = -10; yvec(k) = -10;
        end
    end
end
 xvec = reshape(xvec,lenc*lend,1);
 yvec = reshape(yvec,lenc*lend,1);
    m = find(xvec ~= -10 | yvec ~= -10);
 xvec = xvec(m);
 yvec = yvec(m);
wattr = w;              % This preserves the values of W,Z
zattr = z;
for j = 1:40                                           %
ensures wattr,zattr is
    wattr(j+1) = 1.0 - 1.4*wattr(j)^2 + zattr(j); % on the
attractor
    zattr(j+1) = .3*wattr(j);
        if j > 20 & wattr(j+1) < split, break;
        end
end
 wnew = wattr(j+1);
 znew = zattr(j+1);
wattr = wnew;
zattr = znew;
   xx = xvec;                       % preserves the values of
xvec & yvec
   yy = yvec;
axis([-1.32 1.32 -.6 .5]);
indices = 1:length(xx);           % [1 2 3 4 . . .]
      i = 1;
   qvec = length(xx);
      q = indices;
   lenq = length(xx)
  plot(xvec,yvec,'b.',wattr,zattr,'r*',xline,yline,'w');hold
on;
   plot(o,s,'w.',u,v,'w.',e,f,'w.',g,h,'w.'); hold off;
title('Grid field and point chosen on attractor');pause;clg;
while length(q) > 1;
    axis([-1.32 1.32 -.6 .5]);
    x0 = yy + ones(size(yy)) - 1.4*xx.^2;
    y0 = .3*xx;
    w0 = znew + 1 - 1.4*wnew.^2;
    z0 = .3*wnew;
```

```
         if w0 >= split
            qtemp = find(x0 > split | x0 == split);
         else
            qtemp = find(x0 < split);
         end;
         if length(qtemp) == 0,break;
         else q = qtemp;
         end;
       i = i + 1;           % counts iterates where at least
one point matches
   qvec = [qvec,length(q)];% plot of how # with same binary
sequence
   lenq = length(q)
indices = indices(q);    % decreases each time through loop
     xx = x0(q);         % preserves to next iterate those
matching values of
     yy = y0(q);         % x0 and y0
   wnew = w0;
   znew = z0;
pause;
   plot(xx,yy,'g.',w0,z0,'w+',xline,yline,'w');hold on;
   plot(o,s,'w.',u,v,'w.',e,f,'w.',g,h,'w.');
title('Movement of points in grid with same binary
sequence');
   pause; clg; hold off;
end
axis([1 2 3 4]); axis;
vec = (1/2).^((1:length(qvec))-1);
plot((1:length(qvec))-1,qvec,'r',(1:length(qvec))-1,qvec(1)*
vec,'b');
title('Decrease in # of grid points with same binary
sequence vs 1/2^n')
pause;clg;
% In this way, we need the index key 'indices' to tell to
which grid point the
% surviving iterate corresponds with respect to the original
xvec  and yvec.
axis([-1.32 1.32 -.6 .5]);
plot(xline,yline,'w',x,y,'b.',o,s,'w.',u,v,'w.',e,f,'w.',g,h
,'w.');hold on;
   plot(xvec(indices),yvec(indices),'gx',wattr,zattr,'r*');
%hold on;
title('wattr,zattr & grid pt with same binary
sequence');pause;
hold off;
xnew = xvec(indices);
ynew = yvec(indices); % now we show a plot of how
wattr,zattr and the point
              % that generates the same binary sequence
xnew,ynew
              % walk around the attractor
```

```
axis([-1.32 1.32 -.6 .5]);
plot(x,y,'b.',xline,yline,'w-',o,s,'w.',u,v,'w.',e,f,'w.',g,
h,'w.'); hold on;
 xnewt = xnew;
 ynewt = ynew;
wattrt = wattr;
zattrt = zattr;
for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wattrt.^2 + zattrt;
    z0t = .3*wattrt;
wattrt = w0t;
zattrt = z0t;
 xnewt = x0t;
 ynewt = y0t;
    plot(x0t,y0t,'gx',w0t,z0t,'r*');
title('Movement of pt chosen on HA and matching sequence pt
until diverge');
pause;plot(x0t,y0t,'ix',w0t,z0t,'i*');
end
hold off;
```

```
% function [xmat, ymat] = proof1(spcrs,spfn,incr)
% THIS FUNCTION GOES BACK TO **GRDCOMP1** AND USES IT OVER
MANY SP VALUES.
% This function takes a point(W,Z)from the RT quad,initiates
%a quad grid field based on a certain spacing, then iterates
%those grid field points that match the binary string of
%(W,Z)[use(.1,.1) for now. As long as the strings match,
%points are iterated using the Henon recurrence. Only that
%(those)points that completely match are finally considered.
     w = .5;
     z = .1;
split = .4098;
for sp = spfn:incr:spcrs,                    % MAJOR OUTER LOOP
     sp = sp
      a = [-.1083 -1;-3.64 1;-.1562 1];
      b = [-.2760;-4.6718;-.3344];
      c = [split:sp:1.32];
      d = [-.6:sp:.5];
lenc = length(c);                lend = length(d);
xvec = zeros(1,lenc*lend);     yvec = zeros(1,lenc*lend);
     if min((a*[w;z]) > b) == 0 | w < split
         disp('initial value is not in the right quadr')
         return;
     end
k = 0;
for p = 1:lenc
     for r = 1:lend
         k=k+1;
         if max((a*[c(p);d(r)]) < b) == 0
            xvec(k) = c(p); yvec(k) = d(r);
         else
            xvec(k) = -10; yvec(k) = -10;
         end
     end
end
xvec = reshape(xvec,lenc*lend,1);
yvec = reshape(yvec,lenc*lend,1);
   m = find(xvec ~= -10 | yvec ~= -10);
xvec = xvec(m);  yvec = yvec(m);
xx   = xvec;     yy   = yvec;
ww   = w;        zz   = z;
indices = 1:length(xx);           % [1 2 3 4 . . .]
       i = 1;
   qvec = length(xx);
      q = indices;
   lenq = length(xx);
while length(q) > 1;
     x0 = yy + ones(size(yy))-1.4*xx.^2;
     y0 = .3*xx;
     w0 = zz + 1 - 1.4*ww.^2;
     z0 = .3*ww;
```

```
         if w0 > split
            qtemp = find(x0 > split | x0 == split);
         else
            qtemp = find(x0 < split);
         end;
         if length(qtemp) == 0,break;
         else q = qtemp;
         end
        i = i + 1;                 % counts iterates where at
least one point matches
   lenq = length(q);
   qvec = [qvec,length(q)];% plot of how # with same binary
sequence
indices = indices(q);    % decreases each time through loop
  xx = x0(q);           % preserves to next iterate those
matching values of
   yy = y0(q);           % x0 and y0
   ww = w0;
   zz = z0;
end
% In this way, we need the index key 'indices' to tell to
which grid point the
% surviving iterate corresponds with respect to the original
xvec  and yvec.
xnew = xvec(indices);
ynew = yvec(indices); % now we show a plot of how W,Z and
                     %the point that generates the same
       %binary sequence xnew,ynew walk around the attractor
xnewt  = xnew;
ynewt  = ynew;
    wt = w;
    zt = z;
for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wt.^2 + zt;
    z0t = .3*wt;
      wt = w0t;
      zt = z0t;
  xnewt  = x0t;
  ynewt  = y0t;
end
nvec  = [nvec,n];
spvec = [spvec,sp];
clear xvec;clear yvec;
end                                     % END OF MAJOR LOOP
axis;
   plot(spvec,nvec,'w*');
title('SP VS. ITERATIONS UNTIL DIVERGENCE');
```

```
%  function [xmat, ymat] = proof2(spcrs,spfn,incr)
% THIS FUNCTION GOES BACK TO **GRDCOMP2** AND USES IT OVER
MANY SP VALUES.
% This function takes a (W,Z) from the left quad,initiates a
quad grid field
% based on a certain spacing, then iterates those grid field
points that
% match the binary string of (W,Z) [use (-1,-.25) for now].
As long as the
% strings match, points are iterated using the Henon
recurrence. Only that
% (those) points that completely match are finally
considered.
     w = -1.0;
     z = -.25;
 split = .4098;
for sp = spfn:incr:spcrs,                      % MAJOR OUTER LOOP
    sp = sp
     a = [3.407 1; -.1083 -1; -.1562 1];
     b = [-4.1119; -.2760; -.3344];
     c = [-1.32:sp:split];
     d = [-.6:sp:.5];
lenc = length(c);              lend = length(d);
xvec = zeros(1,lend*lenc);    yvec = zeros(1,lenc*lend);
    if min((a*[w;z]) > b) == 0 | w > split
        disp('initial value is not in the left quadr')
        return;
    end
k=0;
for p = 1:lenc
    for r = 1:lend
        k=k+1;
        if max((a*[c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k) = d(r);
        else
           xvec(k) = -10; yvec(k) = -10;
        end
    end
end
xvec = reshape(xvec,lenc*lend,1);    %makes a column vector
yvec = reshape(yvec,lenc*lend,1);
   m = find(xvec~= -10 | yvec~= -10);
xvec = xvec(m); yvec = yvec(m);
xx   = xvec;    yy   = yvec;
ww   = w;       zz   = z;
indices = 1:length(xx);                    % [1 2 3 4 . . .]
      i = 1;
   qvec = length(xx);
      q = indices;                        % initialization of q
   lenq = length(xx);
while length(q) > 1;
```

```
    x0 = yy + ones(size(yy)) - 1.4*xx.^2;
    y0 = .3*xx;
    w0 = zz + 1 - 1.4*ww.^2;
    z0 = .3*ww;
        if w0 > split
           qtemp = find(x0 > split | x0 == split);
        else
           qtemp = find(x0 < split);
        end;
        if length(qtemp) == 0,break;
        else q = qtemp;
        end
      i = i + 1;
   lenq = length(q);
   qvec = [qvec,length(q)];
indices = indices(q);
     xx = x0(q);       % preserves to next iterate those
matching values of
     yy = y0(q);       % x0 and y0
     ww = w0;
     zz = z0;
end
% In this way, we need the index key 'indices' to tell to
which grid point the
% surviving iterate corresponds with respect to the original
xvec  and yvec.
  xnew = xvec(indices);
  ynew = yvec(indices);
 xnewt = xnew;
 ynewt = ynew;
    wt = w;
    zt = z;
for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wt.^2 + zt;
    z0t = .3*wt;
     wt = w0t;
     zt = z0t;
  xnewt = x0t;
  ynewt = y0t;
end
 nvec = [nvec,n];
spvec = [spvec,sp];
clear xvec;clear yvec;
end                                    % END TO MAJOR OUTER LOOP
   plot(spvec,nvec,'w*');
title('SP VS. ITERATIONS UNTIL DIVERGENCE');
```

```
% function [xmat, ymat] = proof3(spcrs,spfn,incr)
% THIS FXN USES (W,Z)=(-1,-.25). ONE ATTR PT IS COMPARED
OVER MANY SP VALUES
% ONLY DIFF BTWN 3&4 IS THAT 4 WEEDS QUADR,3 USES WHOLE
RECTANGLE.
% This function takes a point(W,Z)from the quadr and
iterates the point n
% times in order to ensure the point is on the Henon
attractor(ensuring
% that the nth iterate is in the left quadrant). It
initiates a grid field
% based on a certain spacing, then iterates those grid field
points that
% match the binary string of (wnew,znew) [use (-1,-.25) for
W,Z]. As long as
% the strings match, points are iterated using the Henon
recurrence. Only that
% (these) points that completely match are finally
considered.
    w = .5;
    z = .1;
split = .4098;
wattr = w;              % This preserves the values of W,Z
zattr = z;
for j = 1:40                                        %
ensures wattr,zattr is
    wattr(j+1) = 1.0 - 1.4*wattr(j)^2 + zattr(j); % on the
attractor
    zattr(j+1) = .3*wattr(j);
        if j > 20 & wattr(j+1) > split, break;
        end
end
wnew  = wattr(j+1);
znew  = zattr(j+1);
for sp = spfn:incr:spcrs,                    % MAJOR OUTER LOOP
    sp = sp
wattr = wnew;
zattr = znew;
    a = [-.1083 -1; -3.64 1; -.1562 1];
    b = [-.2760;-4.6718;-.3344];
    c = [split:sp:1.32];
    d = [-.6:sp:.5];
 lenc = length(c);          lend = length(d);
 xvec = zeros(1,lenc*lend);   yvec = zeros(1,lenc*lend);
    if min((a*[w;z]) > b) == 0 | w < split
        disp('initial value is not in right quadr')
        return;
    end
k=0;
for p = 1:lenc
    for r = 1:lend
```

```
        k=k+1;
   % This next section puts everything outside of the quad
to (-10,-10)
        if max((a * [c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k) = d(r);
        else
           xvec(k) = -10; yvec(k) = -10;
           end
    end
end
xvec = reshape(xvec,lenc*lend,1);
yvec = reshape(yvec,lenc*lend,1);
   m = find(xvec ~= -10 | yvec ~= -10);
xvec = xvec(m);  yvec = yvec(m);
xx   = xvec;     yy   = yvec;

indices = 1:length(xx);          % [1 2 3 4 . . .]
      i = 1;
   qvec = length(xx);
   q    = indices;
   lenq = length(xx);
while length(q) > 1;
    x0 = yy + ones(size(yy)) - 1.4*xx.^2;
    y0 = .3*xx;
    w0 = znew + 1 - 1.4*wnew.^2;
    z0 = .3*wnew;
        if w0 >= split
           qtemp = find(x0 > split | x0 == split);
        else
           qtemp = find(x0 < split);
        end;
        if length(qtemp) == 0,break;
        else q = qtemp;
        end;
      i = i + 1;            % counts iterates where at least
one point matches
   qvec = [qvec,length(q)];   % plot of how # with same
binary sequence
   lenq = length(q);
indices = indices(q);    % decreases each time through loop
    xx = x0(q);          % preserves to next iterate those
matching values of
     yy = y0(q);         % x0 and y0
   wnew = w0;
   znew = z0;
end
% In this way, we need the index key 'indices' to tell to
which grid point the
% surviving iterate corresponds with respect to the original
xvec  and yvec.
xnew = xvec(indices);
```

```
ynew = yvec(indices); % now we show a plot of how
wattr,zattr and the point
              % that generates the same binary sequence
xnew,ynew
              % walk around the attractor
xnewt   = xnew;
ynewt   = ynew;
wattrt  = wattr;
zattrt  = zattr;
for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wattrt.^2 + zattrt;
    z0t = .3*wattrt;
  wattrt = w0t;
  zattrt = z0t;
  xnewt  = x0t;
  ynewt  = y0t;
end
nvec  = [nvec,n];
spvec = [spvec,sp];
clear xvec; clear yvec;
end                                        % END TO MAJOR LOOP
axis;
   plot(spvec,nvec,'w*');
title('SP VS. ITERATIONS UNTIL DIVERGENCE');
```

```
% **function [xmat, ymat] = proof4(spcrs,spfn,incr)**
% THIS FXN USES (W,Z)=(-1,-.25). ONE ATTR PT IS COMPARED
OVER MANY SP VALUES
% ONLY DIFF BTWN 3&4 IS THAT 4 WEEDS QUADR,3 USES WHOLE
RECTANGLE.
% This function takes a point(W,Z)from the quadr and
iterates the point n
% times in order to ensure the point is on the Henon
attractor (ensuring
% that the nth iterate is in the left quadrant). It
initializes a grid field
% based on a certain spacing, then iterates those grid field
points that
% match the binary string of (wnew,znew) [use (-1,-.25) for
W,Z]. As long as
% the strings match, points are iterated using the Henon
recurrence. Only that
%  those  points that completely match are finally
considered.
     w = -1.0;
     z = -.25;
split = .4098;
wattr = w;              % This preserves the values of W,Z
zattr = z;
for j = 1:40                           % ensures wattr,zattr is
     wattr(j+1) = 1.0 - 1.4*wattr(j)^2 + zattr(j); % on the
attractor
     zattr(j+1) = .3*wattr(j);
          if j > 20 & wattr(j+1) < split, break;
          end
end
wnew  = wattr(j+1);
znew  = zattr(j+1);
for sp = spfn:incr:spcrs,                   % MAJOR OUTER LOOP
     sp = sp
wattr = wnew;
zattr = znew;
     a = [3.4074 1; -.1083 -1; -.1562 1];
     b = [-4.1119; -.2760;  -.3344];
     c = [-1.32:sp:split];
     d = [-.6:sp:.5];
  lenc = length(c);          lend = length(d);
  xvec = zeros(1,lenc*lend);   yvec = zeros(1,lenc*lend);
     if min((a*[w;z]) > b) == 0 | w > split
          disp('initial value is not in left quadr')
          return;
     end
k=0;
for p = 1:lenc
     for r = 1:lend
          k=k+1;
```

```
   % This next section puts everything outside of the quad
to (-10,-10)
        if max((a * [c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k) = d(r);
        else
           xvec(k) = -10; yvec(k) = -10;
           end
    end
end
xvec = reshape(xvec,lenc*lend,1);
yvec = reshape(yvec,lenc*lend,1);
   m = find(xvec ~= -10 | yvec ~= -10);
xvec = xvec(m);  yvec = yvec(m);
xx   = xvec;     yy   = yvec;
indices = 1:length(xx);          % [1 2 3 4 . . .]
      i = 1;
   qvec = length(xx);
      q = indices;
   lenq = length(xx);
while length(q) > 1;
    x0 = yy + ones(size(yy)) - 1.4*xx.^2;
    y0 = .3*xx;
    w0 = znew + 1 - 1.4*wnew.^2;
    z0 = .3*wnew;
        if w0 >= split
           qtemp = find(x0 > split | x0 == split);
        else
           qtemp = find(x0 < split);
        end;
        if length(qtemp) == 0,break;
        else q = qtemp;
        end;
      i = i + 1;            % counts iterates where at least
one point matches
   qvec = [qvec,length(q)];   % plot of how # with same
binary sequence
   lenq = length(q);
indices = indices(q);     % decreases each time through loop
     xx = x0(q);          % preserves to next iterate those
matching values of
     yy = y0(q);          % x0 and y0
   wnew = w0;
   znew = z0;
end
% In this way, we need the index key 'indices' to tell to
which grid point the
% surviving iterate corresponds with respect to the original
xvec  and yvec.
xnew = xvec(indices);
```

```
ynew = yvec(indices); % now we show a plot of howwattr,
                      % zattr and the point that generates
                      %the same binary sequence xnew,ynew
                      % walk around the attractor
xnewt  = xnew;
ynewt  = ynew;
wattrt = wattr;
zattrt = zattr;
for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wattrt.^2 + zattrt;
    z0t = .3*wattrt;
  wattrt = w0t;
  zattrt = z0t;
  xnewt  = x0t;
  ynewt  = y0t;
end
nvec  = [nvec,n];
spvec = [spvec,sp];
clear xvec; clear yvec;
end                                     % END TO MAJOR LOOP
axis;
   plot(spvec,nvec,'w*');
title('SP VS. ITERATIONS UNTIL DIVERGENCE');
```

```
% function [xmat, ymat] = prfl(spcrs,spfn,incr)
% This function takes thirty-one points (W,Z) from the RT
quad,initiates a
% quad grid field based on a certain spacing, then iterates
those grid field
% points that match the binary string of (W,Z)[use(.1,.1)
for now. As long as
% the strings match, points are iterated using the Henon
recurrence. Only that
% (those) points that completely match are finally
considered.
   ii = 0;
for w = .5:.02:1.1                          % MAJOR OUTER LOOP
    z = .1;
   ii = ii + 1
split = .4098;
for sp = spfn:incr:spcrs,          % MINOR OUTER LOOP
    sp = sp
     a = [-.1083 -1;-3.64 1;-.1562 1];
     b = [-.2760;-4.6718;-.3344];
     c = [split:sp:1.32];
     d = [-.6:sp:.5];
lenc = length(c);             lend = length(d ;
xvec = zeros(1,lenc*lend);    yvec = zeros(1,lenc*lend);
    if min((a*[w;z]) > b) == 0 | w < split
        disp('initial value is not in the right quadr')
        return;
    end
k = 0;
for p = 1:lenc
    for r = 1:lend
        k=k+1;
        if max((a*[c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k) = d(r);
        else
           xvec(k) = -10; yvec(k) = -10;
        end
    end
end
xvec = reshape(xvec,lenc*lend,1);
yvec = reshape(yvec,lenc*lend,1);
   m = find(xvec ~= -10 | yvec ~= -10);
xvec = xvec(m);  yvec = yvec(m);
xx   = xvec;     yy   = yvec;
ww   = w;        zz   = z;

indices = 1:length(xx);          % [1 2 3 4 . . .]
     .i = 1;
   qvec = length(xx);
      q = indices;
   lenq = length(xx);
```

```
while length(q) > 1;
    x0 = yy + ones(size(yy)) - 1.4*xx.^2;
    y0 = .3*xx;
    w0 = zz + 1 - 1.4*ww.^2;
    z0 = .3*ww;
        if w0 > split
           qtemp = find(x0 > split | x0 == split);
        else
           qtemp = find(x0 < split);
        end;
        if length(qtemp) == 0,break;
        else q = qtemp;
        end
      i = i + 1;              % counts iterates where at
least one point matches
   lenq = length(q);
   qvec = [qvec,length(q)];% plot of how # with same binary
sequence
indices = indices(q);    % decreases each time through loop
   xx = x0(q);          % preserves to next iterate those
matching values of
   yy = y0(q);          % x0 and y0
   ww = w0;
   zz = z0;
end
% In this way, we need the index key 'indices' to tell to
which grid point the
% surviving iterate corresponds with respect to the original
xvec  and yvec.
xnew = xvec(indices);
ynew = yvec(indices); % now we show a plot of how W,Z and
the point
              % that generates the same binary sequence
xnew,ynew
              % walk around the attractor
xnewt   = xnew;
ynewt   = ynew;
    wt = w;
    zt = z;
for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wt.^2 + zt;
    z0t = .3*wt;
      wt = w0t;
      zt = z0t;
  xnewt   = x0t;
  ynewt   = y0t;
end
nvec1  = [nvec1,n];
spvec1 = [spvec1,sp];
```

```
clear xvec;clear yvec;
end                                            % END MINOR LOOP
nvec2  = [nvec2,nvec1];
spvec2 = [spvec2,spvec1];
clear xvec; clear yvec;
end                                            % END MAJOR LOOP
axis([0 max(spvec2) 0 max(nvec2)]);
   plot(spvec2,nvec2,'w.');
title('SP VS. ITERATIONS UNTIL DIVERGENCE');
```

```
% function [xmat, ymat] = prf2(spcrs,spfn,incr)
% THIS FUNCTION GOES BACK TO **GRDCOMP2** AND USES IT OVER
MANY SP VALUES.
% (AND OVER THIRTY-ONE W,Z VALUES)
% This function takes a (W,Z) from the left quad,initiates a
quad grid field
% based on a certain spacing, then iterates those grid field
points that
% match the binary string of (W,Z) [use (-1,-.25) for now].
As long as the
% strings match, points are iterated using the Henon
recurrence. Only that
% (those) points that completely match are finally
considered.
    ii = 0;
for  w = -.80:.04:.4                          % MAJOR OUTER LOOP
      z = -.01
     ii = ii + 1
  split = .4098;
    for sp = spfn:incr:spcrs,             % MINOR OUTER LOOP
     sp = sp
      a = [3.407 1; -.1083 -1; -.1562 1];
      b = [-4.1119; -.2760; -.3344];
      c = [-1.32:sp:split];
      d = [-.6:sp:.5];
lenc = length(c);               lend = length(d);
xvec = zeros(1,lend*lenc);     yvec = zeros(1,lenc*lend);
     if min((a*[w;z]) > b) == 0 | w > split
         disp('initial value is not in the left quadr')
         return;
     end
k=0;
for p = 1:lenc
     for r = 1:lend
         k=k+1;
         if max((a*[c(p);d(r)]) < b) == 0
            xvec(k) = c(p); yvec(k) = d(r);
         else
            xvec(k) = -10; yvec(k) = -10;
         end
     end
end
xvec = reshape(xvec,lenc*lend,1);     %makes a column vector
yvec = reshape(yvec,lenc*lend,1);
   m = find(xvec~= -10 | yvec~= -10);
xvec = xvec(m); yvec = yvec(m);
xx   = xvec;    yy   = yvec;
ww   = w;       zz   = z;
indices = 1:length(xx);                       % [1 2 3 4 . . .]
      i = 1;
   qvec = length(xx);
```

```
      q = indices;                                    %
initialization of q
   lenq = length(xx);
while length(q) > 1;
    x0 = yy + ones(size(yy)) - 1.4*xx.^2;
    y0 = .3*xx;
    w0 = zz + 1 - 1.4*ww.^2;
    z0 = .3*ww;
        if w0 > split
           qtemp = find(x0 > split | x0 == split);
        else
           qtemp = find(x0 < split);
        end;
        if length(qtemp) == 0,break;
        else q = qtemp;
        end
      i = i + 1;
   lenq = length(q);
   qvec = [qvec,length(q)];
indices = indices(q);
     xx = x0(q);        % preserves to next iterate those
matching values of
     yy = y0(q);        % x0 and y0
     ww = w0;
     zz = z0;
end
% In this way, we need the index key 'indices' to tell to
which grid point the
% surviving iterate corresponds with respect to the original
xvec  and yvec.
  xnew = xvec(indices);
  ynew = yvec(indices);
 xnewt = xnew;
 ynewt = ynew;
    wt = w;
    zt = z;
for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wt.^2 + zt;
    z0t = .3*wt;
     wt = w0t;
     zt = z0t;
  xnewt = x0t;
  ynewt = y0t;
end

    nvec1 = [nvec1,n];
   spvec1 = [spvec1,sp];
clear xvec;clear yvec;
   end                                  % END TO MINOR OUTER LOOP
 nvec2 = [nvec2,nvec1];
spvec2 = [spvec2,spvec1];
end                              % END TO MAJOR OUTER LOOP
axis([0 max(spvec2) 0 max(nvec2)]);
   plot(spvec2,nvec2,'w.');
title('SP VS. ITERATIONS UNTIL DIVERGENCE');
```

```
% function [xmat, ymat] = prf3(spcrs,spfn,incr)
% THIS FXN USES THIRTY-ONE W,Z VALUES. ONE ATTR PT IS
COMPARED OVER MANY SP VALUES
% ONLY DIFF BTWN 3&4 IS THAT 4 USES LEFT QUADR , 3 USES
RIGHT QUADR.
% This function takes a pointS (W,Z) from the quadr and
iterates the point n
% times in order to ensure the point is on the Henon
attractor(ensuring
% that the nth iterate is in the left quadrant). It
initiates a grid field
% based on a certain spacing, then iterates those grid field
points that
% match the binary string of (wnew,znew) [use (-1,-.25) for
W,Z]. As long as
% the strings match, points are iterated using the Henon
recurrence. Only that
% (those) points that completely match are finally
considered.
   ii = 0;
for w = .5:.02:1.1                          % MAJOR OUTER LOOP
     z = .1;
   ii = ii + 1
split = .4098;
wattr = w;              % This preserves the values of W,Z
zattr = z;
for j = 1:40            % ensures wattr,zattr is
    wattr(j+1) = 1.0 - 1.4*wattr(j)^2 + zattr(j); % on the
attractor
    zattr(j+1)' = .3*wattr(j);
        if j > 20 & wattr(j+1) > split, break;
        end
end
wnew  = wattr(j+1);
znew  = zattr(j+1);
for sp = spfn:incr:spcrs,                  % MINOR OUTER LOOP
    sp = sp
wattr = wnew;
zattr = znew;
    a = [-.1083 -1; -3.64 1; -.1562 1];
    b = [-.2760;-4.6718;-.3344];
    c = [split:sp:1.32];
    d = [-.6:sp:.5];
 lenc = length(c);             lend = length(d);
 xvec = zeros(1,lenc*lend);   yvec = zeros(1,lenc*lend);
    if min((a*[w;z]) > b) == 0 | w < split
        disp('initial value is not in right quadr')
        return;
    end
k=0;
for p = 1:lenc
```

```
    for r = 1:lend
        k=k+1;
   % This next section puts everything outside of the quad
to (-10,-10)
        if max((a * [c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k) = d(r);
        else
           xvec(k) = -10; yvec(k) = -10;
           end
    end
end
xvec = reshape(xvec,lenc*lend,1);
yvec = reshape(yvec,lenc*lend,1);
   m = find(xvec ~= -10 | yvec ~= -10);
xvec = xvec(m);  yvec = yvec(m);
xx   = xvec;     yy   = yvec;
indices = 1:length(xx);          % [1 2 3 4 . . .]
      i = 1;
   qvec = length(xx);
   q    = indices;
   lenq = length(xx);
while length(q) > 1;
    x0 = yy + ones(size(yy)) - 1.4*xx.^2;
    y0 = .3*xx;
    w0 = znew + 1 - 1.4*wnew.^2;
    z0 = .3*wnew;
        if w0 >= split
           qtemp = find(x0 > split | x0 == split);
        else
           qtemp = find(x0 < split);
        end;
        if length(qtemp) == 0,break;
        else q = qtemp;
        end;
      i = i + 1;            % counts iterates where at least
one point matches
   qvec = [qvec,length(q)];   % plot of how # with same
binary sequence
   lenq = length(q);
indices = indices(q);    % decreases each time through loop
    xx = x0(q);          % preserves to next iterate those
matching values of
     yy = y0(q);          % x0 and y0
   wnew = w0;
   znew = z0;
end
% In this way, we need the index key 'indices' to tell to
which grid point the
% surviving iterate corresponds with respect to the original
xvec  and yvec.
xnew = xvec(indices);
```

```
ynew = yvec(indices); % now we show a plot of how
wattr,zattr and the point
                 % that generates the same binary sequence
xnew,ynew
                 % walk around the attractor
xnewt  = xnew;
ynewt  = ynew;
wattrt = wattr;
zattrt = zattr;
for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wattrt.^2 + zattrt;
    z0t = .3*wattrt;
  wattrt = w0t;
  zattrt = z0t;
  xnewt  = x0t;
  ynewt  = y0t;
end                                         % END TO MINOR LOOP
nvec1  = [nvec1,n];
spvec1 = [spvec1,sp];
clear xvec; clear yvec;
end                                        % END TO MAJOR LOOP
nvec2  = [nvec2,nvec1];
spvec2 = [spvec2,spvec1];
end
axis([0 max(spvec2) 0 max(nvec2)]);
   plot(spvec2,nvec2,'.');
title('SP VS. ITERATIONS UNTIL DIVERGENCE');
```

```
% function [xmat, ymat] = prf4(spcrs,spfn,incr)
% THIS FXN USES THIRTY-ONE W,Z VALUES. ONE ATTR PT IS
COMPARED OVER MANY SP VALUES
% ONLY DIFF BTWN 3&4 IS THAT 4 USES THE LEFT QUADR, 3 USES
THE RIGHT QUADR.
% This function takes thirty points(W,Z)from the quad and
iterates the point n
% times in order to ensure the point is on the Henon
attractor(ensuring
% that the nth iterate is in the left quadrant). It
initiates a grid field
% based on a certain spacing, then iterates those grid field
points that
% match the binary string of (wnew,znew) [use (-1,-.25) for
W,Z]. As long as
% the strings match, points are iterated using the Henon
recurrence. Only that
% (those) points that completely match are finally
considered.
   ii = 0;
for w = -.80:.04:.4                 % MAJOR OUTER LOOP
    z = -.01
   ii = ii + 1
split = .4098;
wattr = w;              % This preserves the values of W,Z
zattr = z;
for j = 1:40                                       % ensures
                                                  %wattr,zattr is
    wattr(j+1) = 1.0 - 1.4*wattr(j)^2 + zattr(j); % on the
                                                    %attractor
    zattr(j+1) = .3*wattr(j);
        if j > 20 & wattr(j+1) < split, break;
        end
end
wnew  = wattr(j+1);
znew  = zattr(j+1);
for sp = spfn:incr:spcrs,                        % MINOR OUTER LOOP
    sp = sp
wattr = wnew;
zattr = znew;
    a = [3.4074 1; -.1083 -1; -.1562 1];
    b = [-4.1119; -.2760;  -.3344];
    c = [-1.32:sp:split];
    d = [-.6:sp:.5];
 lenc = length(c);            lend = length(d);
 xvec = zeros(1,lenc*lend);   yvec = zeros(1,lenc*lend);
    if min((a*[w;z]) > b) == 0 | w > split
        disp('initial value is not in left quadr')
        return;
    end
k=0;
```

```
for p = 1:lenc
    for r = 1:lend
        k=k+1;
   % This next section puts everything outside of the quad
to (-10,-10)
        if max((a * [c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k) = d(r);
        else
           xvec(k) = -10; yvec(k) = -10;
           end
    end
end
xvec = reshape(xvec,lenc*lend,1);
yvec = reshape(yvec,lenc*lend,1);
   m = find(xvec ~= -10 | yvec ~= -10);
xvec = xvec(m);  yvec = yvec(m);
xx   = xvec;     yy   = yvec;
indices = 1:length(xx);          % [1 2 3 4 . . .]
      i = 1;
   qvec = length(xx);
      q = indices;
   lenq = length(xx);
while length(q) > 1;
    x0 = yy + ones(size(yy)) - 1.4*xx.^2;
    y0 = .3*xx;
    w0 = znew + 1 - 1.4*wnew.^2;
    z0 = .3*wnew;
        if w0 >= split
           qtemp = find(x0 > split | x0 == split);
        else
           qtemp = find(x0 < split);
        end;
        if length(qtemp) == 0,break;
        else q = qtemp;
        end;
      i = i + 1;           % counts iterates where at least
one point matches
   qvec = [qvec,length(q)];   % plot of how # with same
binary sequence
   lenq = length(q);
indices = indices(q);    % decreases each time through loop
     xx = x0(q);         % preserves to next iterate those
matching values of
     yy = y0(q);         % x0 and y0
   wnew = w0;
   znew = z0;
end
% In this way, we need the index key 'indices' to tell to
which grid point the
% surviving iterate corresponds with respect to the original
xvec  and yvec.
```

```
xnew = xvec(indices);
ynew = yvec(indices); % now we show a plot of how
wattr,zattr and the point
                 % that generates the same binary sequence
xnew,ynew
                 % walk around the attractor
xnewt  = xnew;
ynewt  = ynew;
wattrt = wattr;
zattrt = zattr;
for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wattrt.^2 + zattrt;
    z0t = .3*wattrt;
  wattrt = w0t;
  zattrt = z0t;
  xnewt  = x0t;
  ynewt  = y0t;
end
nvec1  = [nvec1,n];
spvec1 = [spvec1,sp];
clear xvec; clear yvec;
end                                  % END TO MINOR LOOP
nvec2  = [nvec2,nvec1];
spvec2 = [spvec2,spvec1];
end                                  % END TO MAJOR LOOP
axis([0 max(spvec2) 0 max(nvec2)]);
   plot(spvec2,nvec2,'w.');
title('SP VS. ITERATIONS UNTIL DIVERGENCE');
```

```
% ONTO.M
% THIS PROGRAM REQUIRES THE PROGRAM HENON TO RUN PROPERLY
% This program is a derivative of a program used originally
% in reference 13 called "runs.m".
% This program is run using the word-"onto"
% To run use "x = henon(#,0,0,)"; then use "onto";
% for the appropriate n-tuple sought.
x = henon(500000,0,0);
n = 4
x = x';
%decvec2 = zeros(4,1);
%decvec3 = zeros(8,1);
decvec4 = zeros(16,1);
%decvec5 = zeros(32,1);
%decvec6 = zeros(64,1);
%decvec7 = zeros(128,1);
%decvec8 = zeros(256,1);
%decvec9 = zeros(512,1);
%decvec10 = zeros(1024,1);
%decvec11 = zeros(2048,1);
%decvec12 = zeros(4096,1);
%decvec13 = zeros(8192,1);
%decvec14 = zeros(16384,1);
%decvec15 = zeros(32768,1);
%decvec16 = zeros(65536,1);
%decvec17 = zeros(131072,1);
 i = 0;

 while min(decvec4)== 0
        i = i + 1;

%decimal = [2 1] * x(i:i+1);
%decvec2(decimal+1) = decvec2(decimal+1) + 1;

%decimal=[4 2 1]*x(i:i+2);
%decvec3(decimal+1) = decvec3(decimal+1) + 1;

decimal=[8 4 2 1]*x(i:i+3);
decvec4(decimal+1) = decvec4(decimal+1) + 1;

%decimal=[16 8 4 2 1]*x(i:i+4);
%decvec5(decimal+1) = decvec5(decimal+1) + 1;

%decimal=[32 16 8 4 2 1]*x(i:i+5);
%decvec6(decimal+1) = decvec6(decimal+1) + 1;

%decimal=[64 32 16 8 4 2 1]*x(i:i+6);
%decvec7(decimal+1) = decvec7(decimal+1) + 1;

%decimal=[128 64 32 16 8 4 2 1]*x(i:i+7);
%decvec8(decimal+1) = decvec8(decimal+1) + 1;
```

```
%decimal=[256 128 64 32 16 8 4 2 1 ]*x(i:i+8);
%decvec9(decimal+1) = decvec9(decimal+1) + 1;

%decimal=[512 256 128 64 32 16 8 4 2 1]*x(i:i+9);
%decvec10(decimal+1) = decvec10(decimal+1) + 1;

%decimal=[1024 512 256 128 64 32 16 8 4 2 1]*x(i:i+10);
%decvec11(decimal+1) = decvec11(decimal+1) + 1;

%decimal=[2048 1024 512 256 128 64 32 16 8 4 2 1]*x(i:i+11);
%decvec12(decimal+1) = decvec12(decimal+1) + 1;

%decimal=[4096 2048 1024 512 256 128 64 32 16 8 4 2
1]*x(i:i+12);
%decvec13(decimal+1) = decvec13(decimal+1) + 1;

%v=[8192 4096 2048 1024 512 256 128 64 32 16 8 4 2 1];
%decimal=v*x(i:i+13);
%decvec14(decimal+1) = decvec14(decimal+1) + 1;

%v=[16384 8192 4096 2048 1024 512 256 128 64 32 16 8 4 2 1];
%decimal= v * x(i:i+14);
%decvec15(decimal+1) = decvec15(decimal+1) + 1;

%v=[32768 16384 8192 4096 2048 1024 512 256 128 64 32 16 8 4
2 1];
%decimal = v * x(i:i+15);
%decvec16(decimal+1) = decvec16(decimal+1) + 1;

%v=[65536 32768 16384 8192 4096 2048 1024 512 256 128 64];
%decimal=[v 32 16 8 4 2 1] * x(i:i+16);
%decvec17(decimal+1) = decvec17(decimal+1) + 1;

end

% i tells the sequence length at which all the n-tuples were
found.

n % n-tuple looking for
i % length of decvec where found
length(x) % total length of original vector from Henon
```

```
%function xx = find110(n,x0,y0)
% This program requires that plot(x110,y110,'.') be used
%after the run. This program takes n points on the attractor
%and finds those that give the sequence 110.  It then
%iterates the set 3 times.  Merely set the parameters below
%and type "find110".
  n = 5000;
 x0 = 0;
 y0 = 0;
%inputs:
%       n = length of desired sequence
%      x0 = initial x
%      y0 = initial y

%outputs:
%      xx = n by 1 binary vector
 hold on;

x(1) = x0;
y(1) = y0;

o=linspace(-1.33,1.32,500);
s=-.1083*o + .276;
u=linspace(1.32,1.245,500);
v=3.64*u - 4.6718;
g=linspace(-1.06,1.245,500);
h=.1533*g - .3344;
e=linspace(-1.06,-1.33,500);
f=-3.407*e - 4.1119;

split = .4098;
axis([-1.32 1.32 -.6 .5]);
xline = [split split]; yline = [-1.32 1.32];
plot(xline,yline,'w-');hold on;
plot(o,s,'w.',u,v,'w.',e,f,'w.',g,h,'w.');
title('Exclusion Area of Points which Never Give Sequence
1100')
pause;

x(2:n) = zeros(n-1,1);   % vectors are preallocated
y(2:n) = zeros(n-1,1);

%recursive generation of points
for i = 1:n;
        x(i+1) = y(i) + 1 - 1.4*x(i)^2;
        y(i+1) = .3 * x(i);

% convert to binary
        if x(i) <= split
                xx(i) = 0;
        else
```

```
                    xx(i) = 1;
            end
    end

    %      xx
    decvec3 = zeros(8,1);
          i = 0;

       while min(decvec3)==0
                  i = i + 1;
                  decimal = [4 2 1]*xx(i:i+2)';
                        if decimal==6,     x110 = [x110,x(i)];
                                           y110 = [y110,y(i)];
             end
       end

    %    plot(x110,y110,'.')
```

```
% SUBQUAD
% This program shows how the four sides of the
subquadrilateral
% shift as the sides are iterated 3 times. This ensures that
any pt.
% falling in the subquad will not give the sequence 1100.
% To run type "subquad"
xx = linspace(-1.33,1.32,500);
yy = -.1083*xx + .2760;

aa = linspace(1.32,1.245,500);
bb = 3.64*aa - 4.6718;

cc = linspace(-1.06,1.245,500);
dd = .1533*cc -.3344;

ee = linspace(-1.06,-1.33,500);
ff = -3.407*ee - 4.1119;

x = linspace(.4099,.5714,500);
y = -.14*x + .2785;

a = linspace(.5714,.5273,500);
b = 1.769*a - .8123;

c = linspace(.5273,.4102,500);
d = -.2673*c + .26145;

e = linspace(.4102, .4099,500);
f = -232.7*e + 95.6;

split = .4098;
xline = [split split];
yline = [-1.32 1.32];

axis([-1.32 1.32 -.6 .5]);
hold on;plot(xline,yline,'g-');                         % split
plot(xx,yy,'g.',aa,bb,'g.',cc,dd,'g.',ee,ff,'g.'); %
quadrilateral
plot(x,y,'w.',a,b,'w.',c,d,'w.',e,f,'w.');              % subquad
title('Subquad,Quadrilateral and Split of.4098');
disp('about to print')
print;
pause;clg;
hold off

%while s ~= 0

for iter=1:3

sprintf('Iter=%g',iter)
```

```
    for n = 1:500
     x0(n) = 1.0 - 1.4*x(n)^2 + y(n);
     y0(n) = .3*x(n);
    end
                              % 1
    for i = 1:500
     a0(i) = 1.0 - 1.4*a(i)^2 + b(i);
     b0(i) = .3*a(i);
    end
                              % 2
    for j = 1:500
     c0(j) = 1.0 - 1.4*c(j)^2 + d(j);
     d0(j) = .3*c(j);
    end
                              % 3
    for k = 1:500
     e0(k) = 1.0 - 1.4*e(k)^2 + f(k);
     f0(k) =  .3*e(k);
    end
                              % 4
axis([-1.32 1.32 -.6 .5]);

%s = s + 1;
%subplot(221);
hold on;plot(xline,yline,'g-');                          % split
plot(xx,yy,'g.',aa,bb,'g.',cc,dd,'g.',ee,ff,'g.'); %
quadrilateral
plot(x0,y0,'w.',a0,b0,'w.',c0,d0,'w.',e0,f0,'w.'); %
iterated subquad
title('Iterated Subquad,Quadrilateral and Split of .4098');
pause;
hold off

%Prepare for next iteration.
x=x0;y=y0;a=a0;b=b0;c=c0;d=d0;e=e0;f=f0;
%   if s==3
    print;clg
    end
end;
```

APPENDIX B: TYPICAL LEFT QUADRILATERAL RUN

The following is a typical run of GRDCOMP6.M which models which points from the left quadrilateral give the same binary sequence as a point (W,Z) chosen at random from the left quadrilateral. The point (W,Z) (+) is indicated with a long arrow; (u,v) points (x) are indicated with shorter arrows.

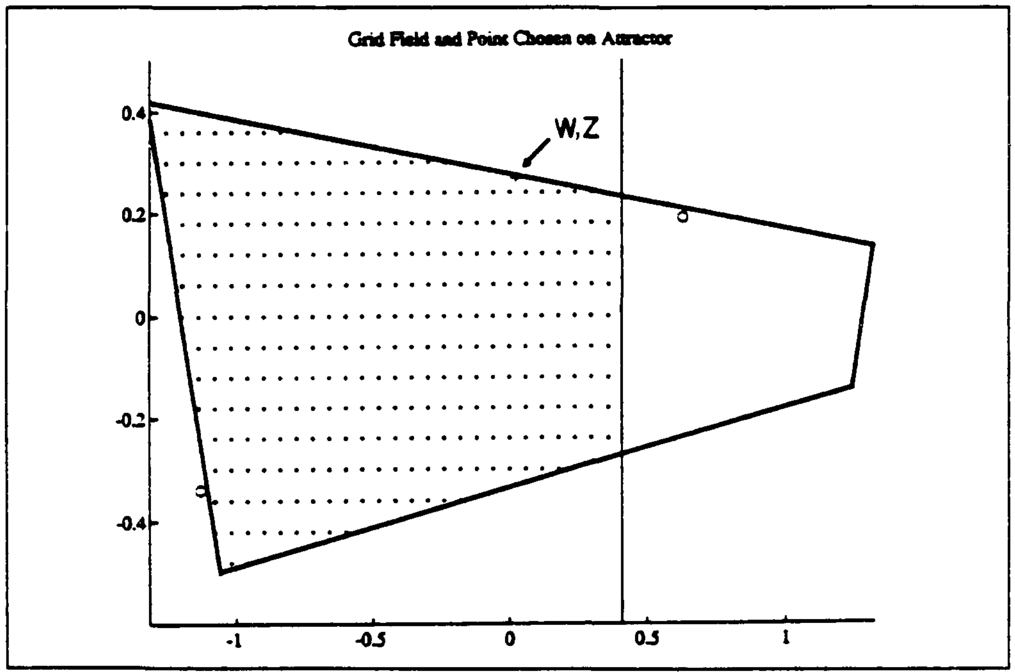

Figure 55 Typical run left quad model: before iteration

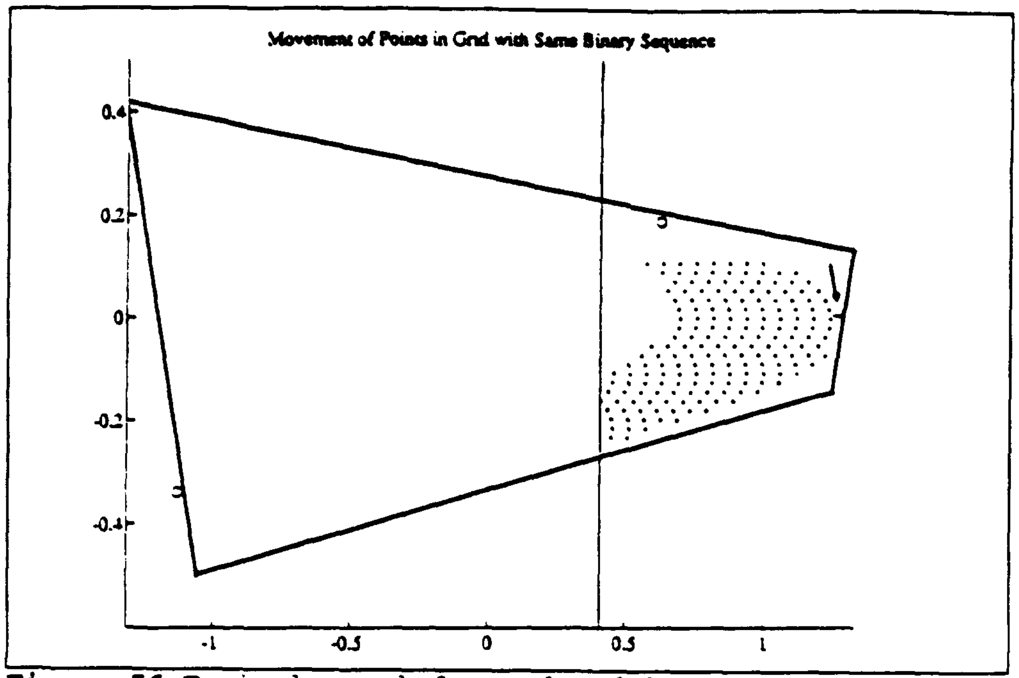

Figure 56 Typical run left quad model: Iterate 1

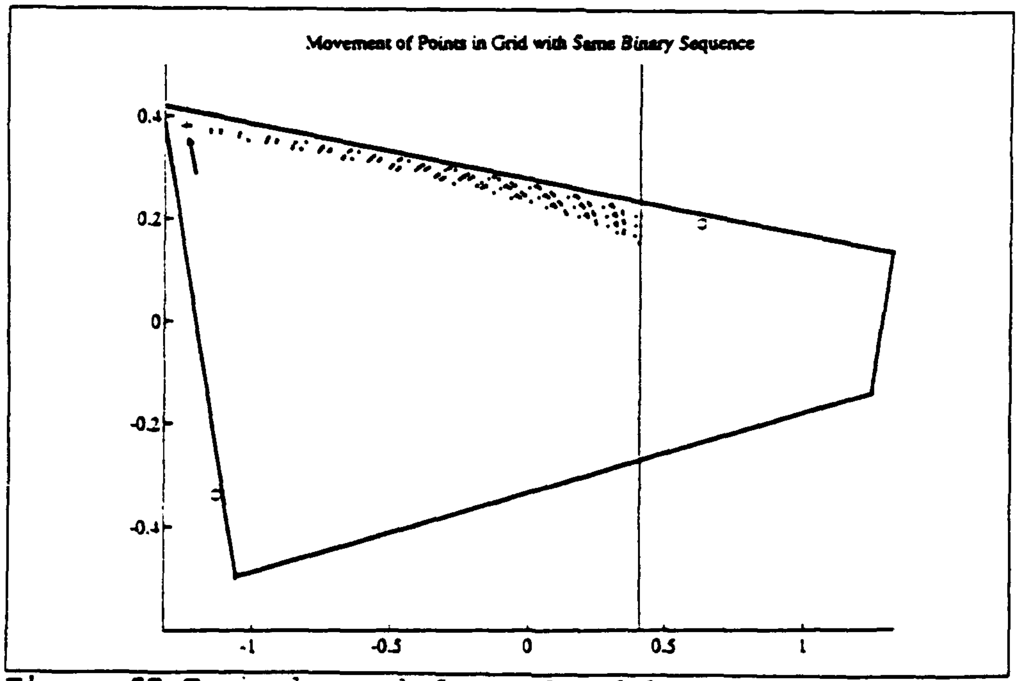

Figure 57 Typical run left quad model: Iterate 2

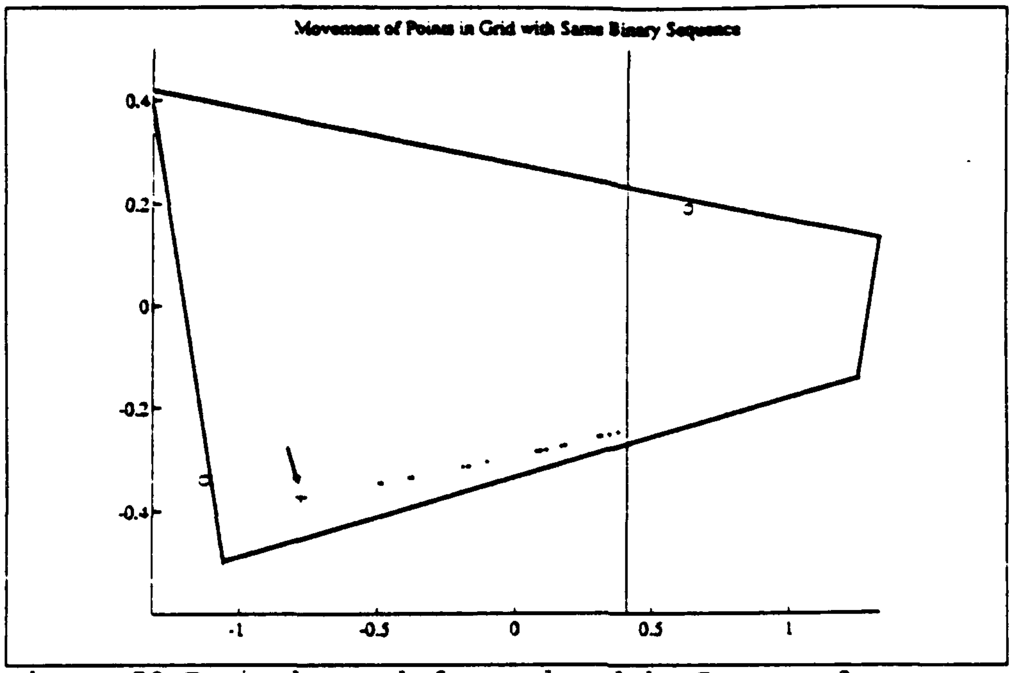

Figure 58 Typical run left quad model: Iterate 3

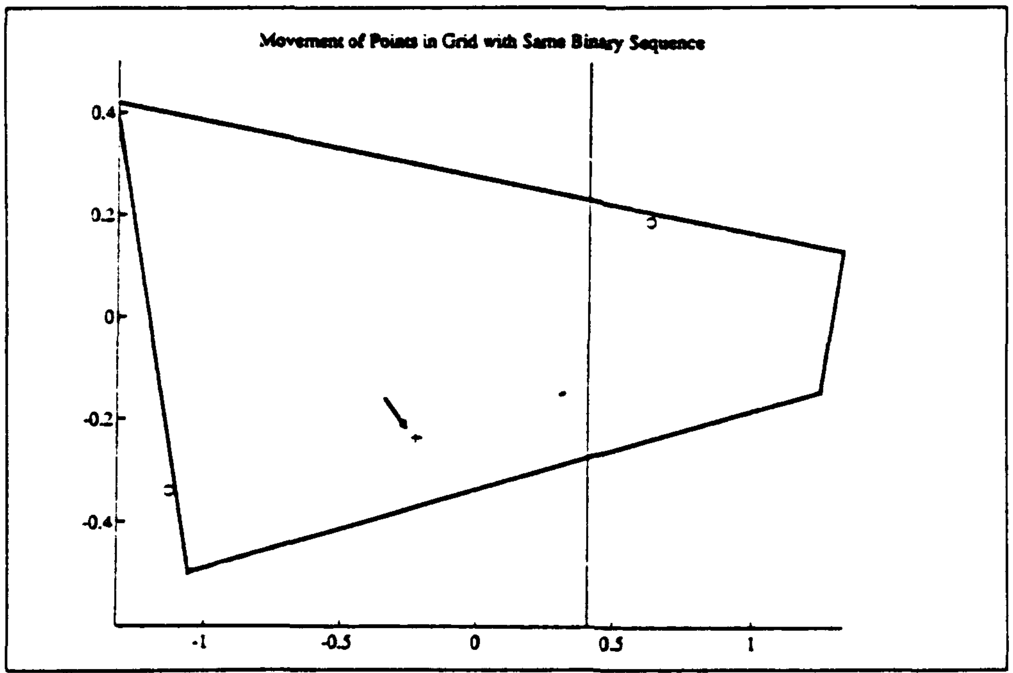

Figure 59 Typical run left quad model: Iterate 4

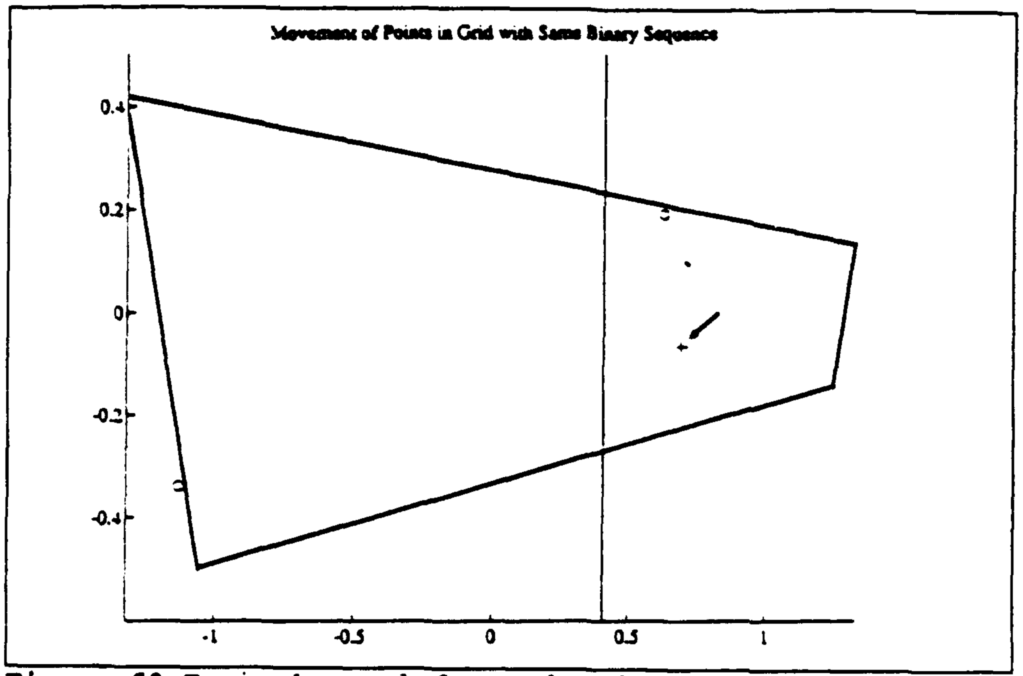

Figure 60 Typical run left quad model: Iterate 5

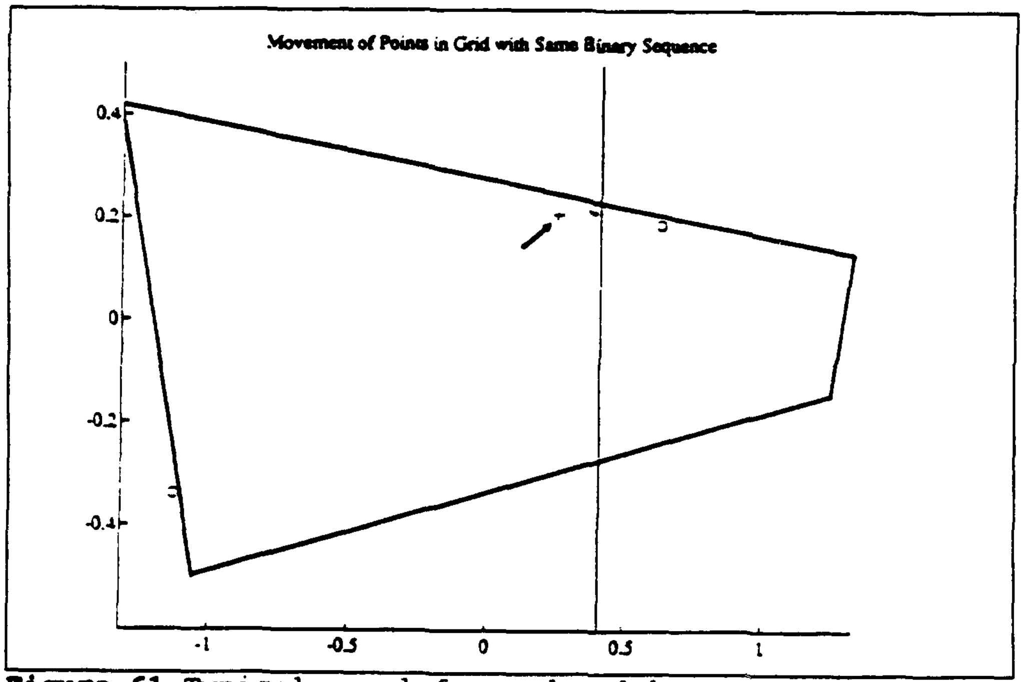

Figure 61 Typical run left quad model: Iterate 6

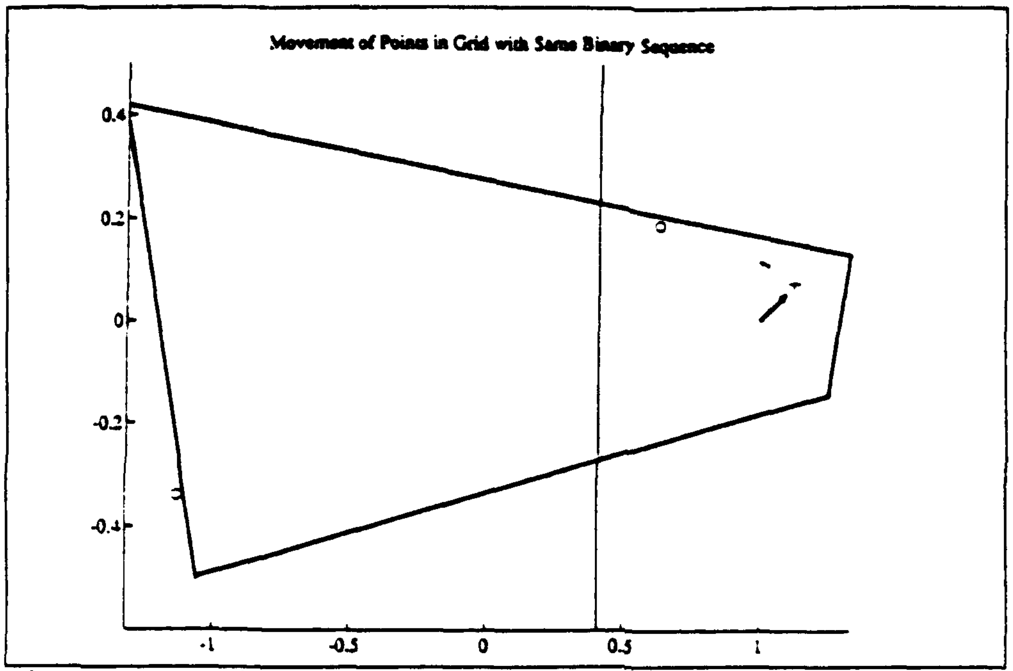

Figure 62 Typical run left quad model: Iterate 7

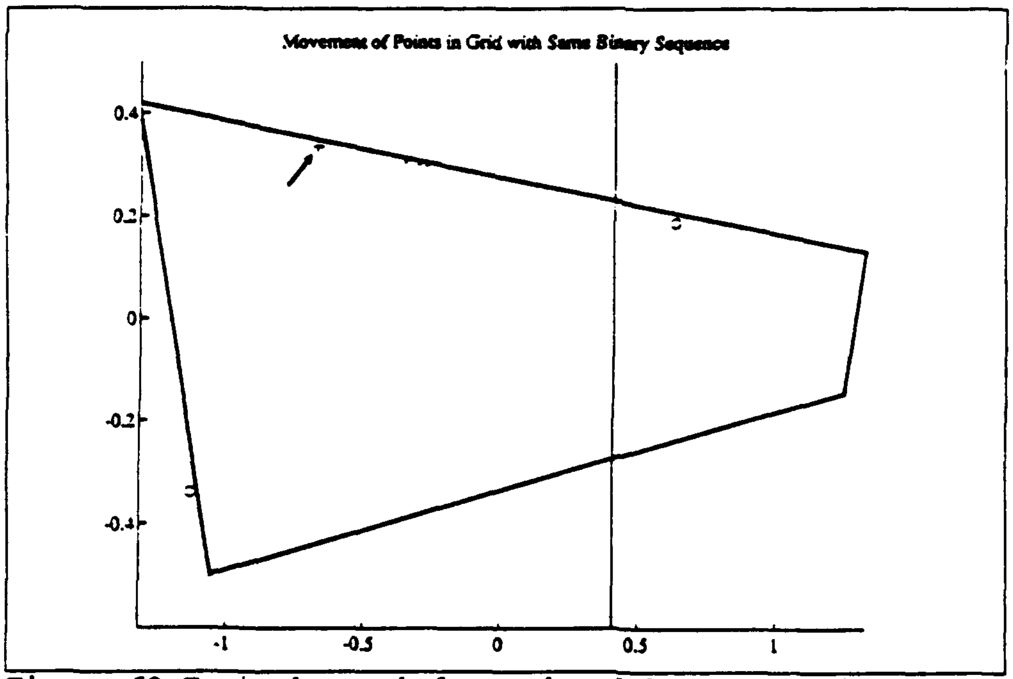

Figure 63 Typical run left quad model: Iterate 8

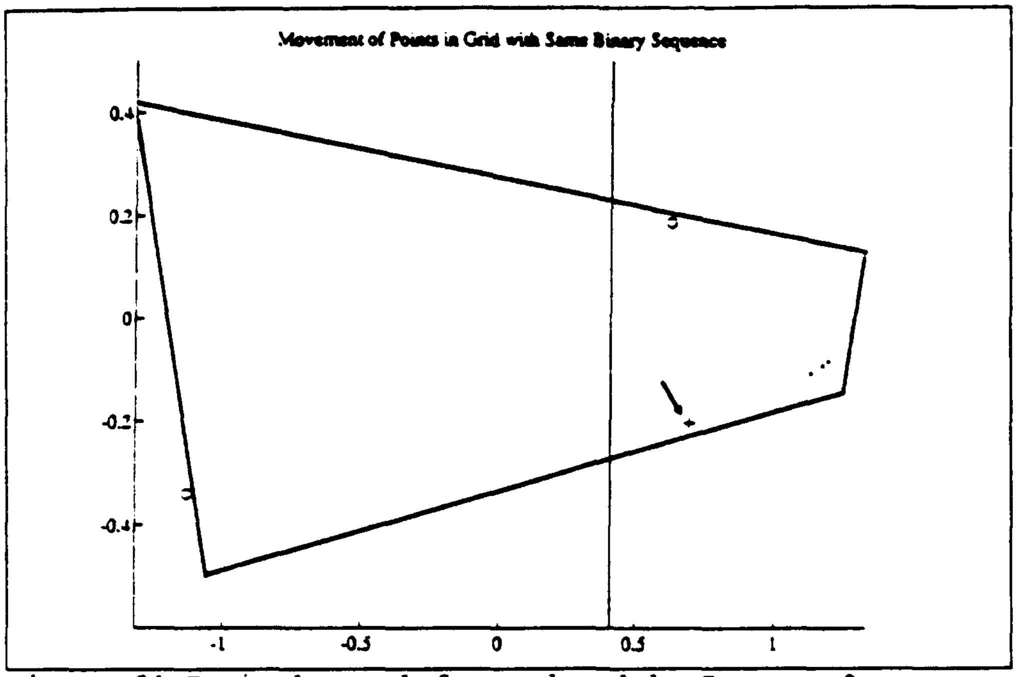

Figure 64 Typical run left quad model: Iterate 9

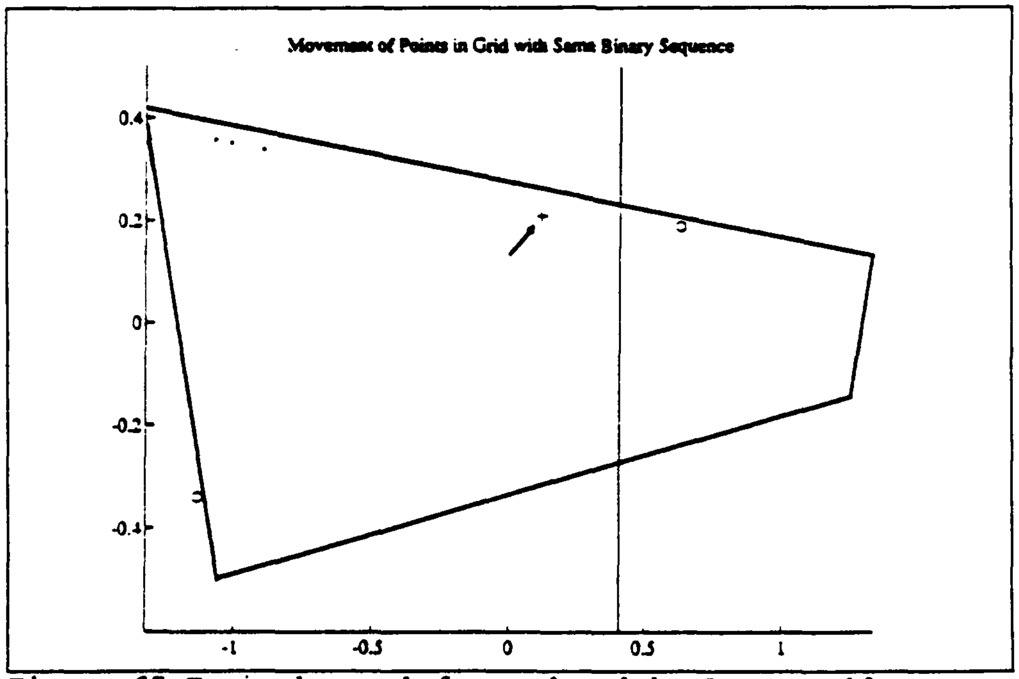

Figure 65 Typical run left quad model: Iterate 10

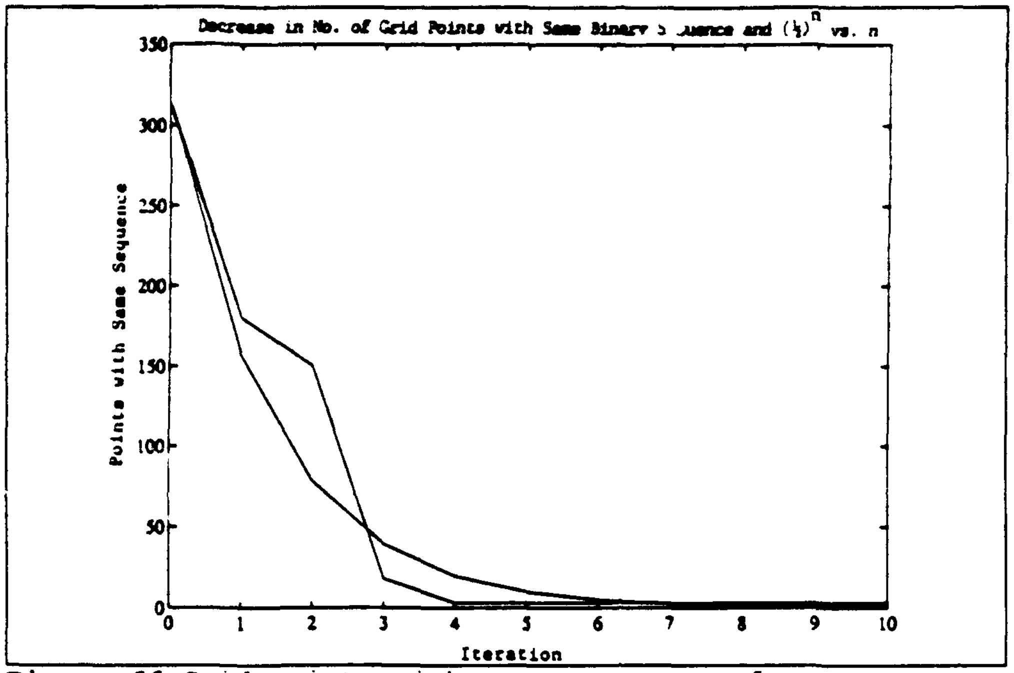

Figure 66 Grid points with same sequence after n iterations

Table 5 shows the count of points for iterates 1 through 10 that give the same binary sequence as (W,Z). At iterate 11 (not shown) there are no grid points that give the same binary sequence as (W,Z).

TABLE 5 NUMBER OF POINTS THAT GIVE THE SAME BINARY SEQUENCE

NUMBER OF POINTS THAT GIVE THE SAME BINARY SEQUENCE											
ITERATE	0	1	2	3	4	5	6	7	8	9	10
NO. PTS	314	180	150	18	3	3	3	3	3	3	3

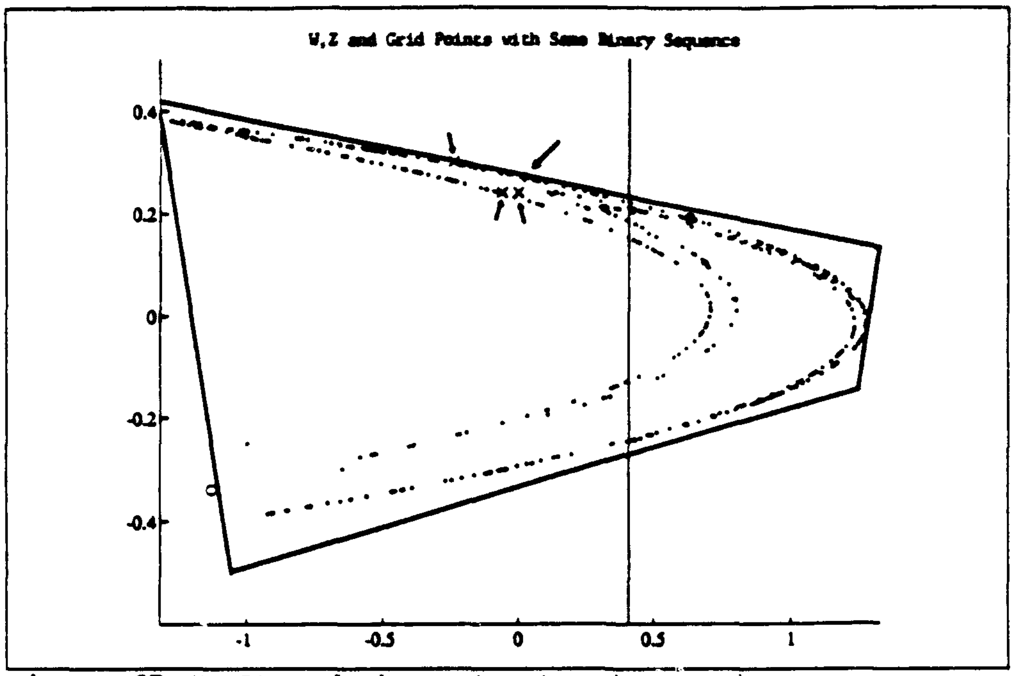

Figure 67 (W,Z) and three (u,v) points prior to iteration

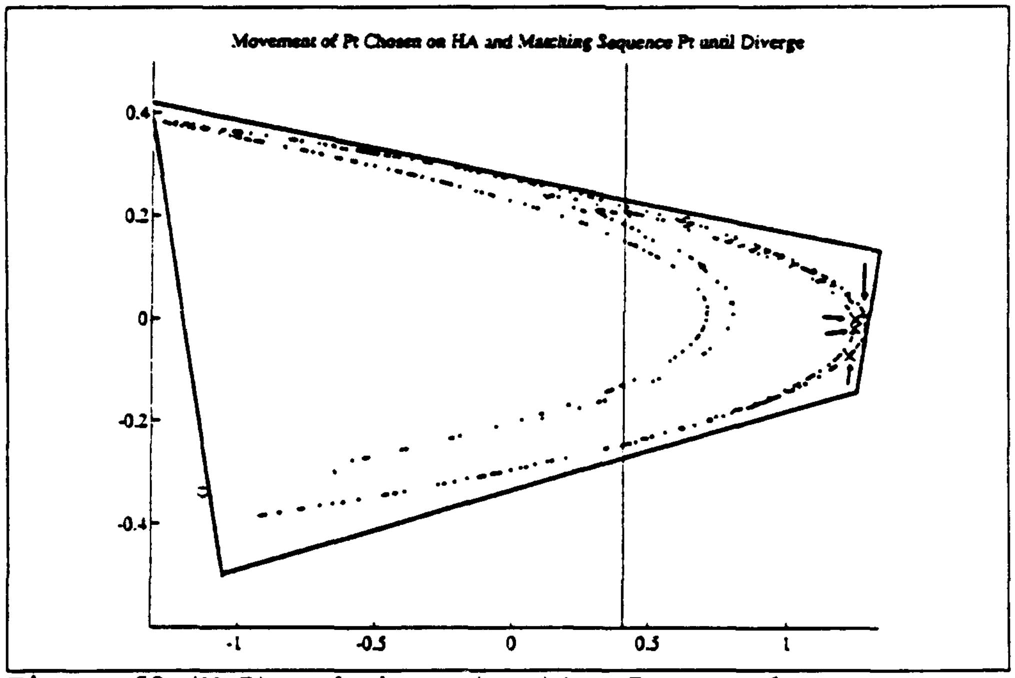

Figure 68 (W,Z) and three (u,v)'s: Iterate 1

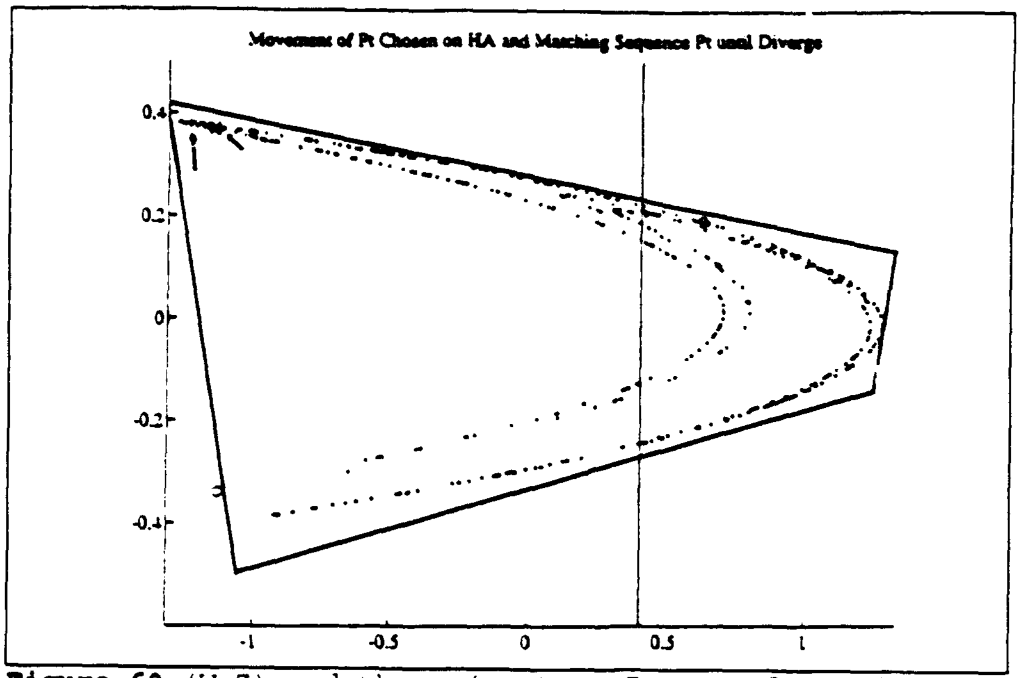

Figure 69 (W,Z) and three (u,v)'s: Iterate 2 (Notice that the (u,v) points are essentially superimposed).

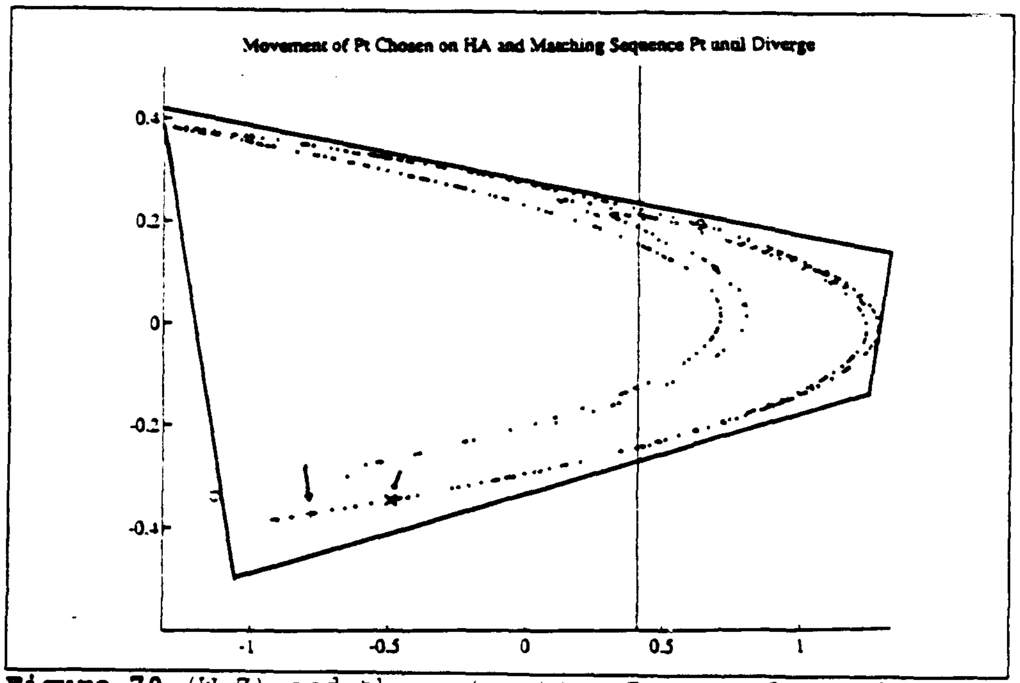

Figure 70 (W,Z) and three (u,v)'s: Iterate 3 (Notice that the (u,v) points are essentially superimposed).

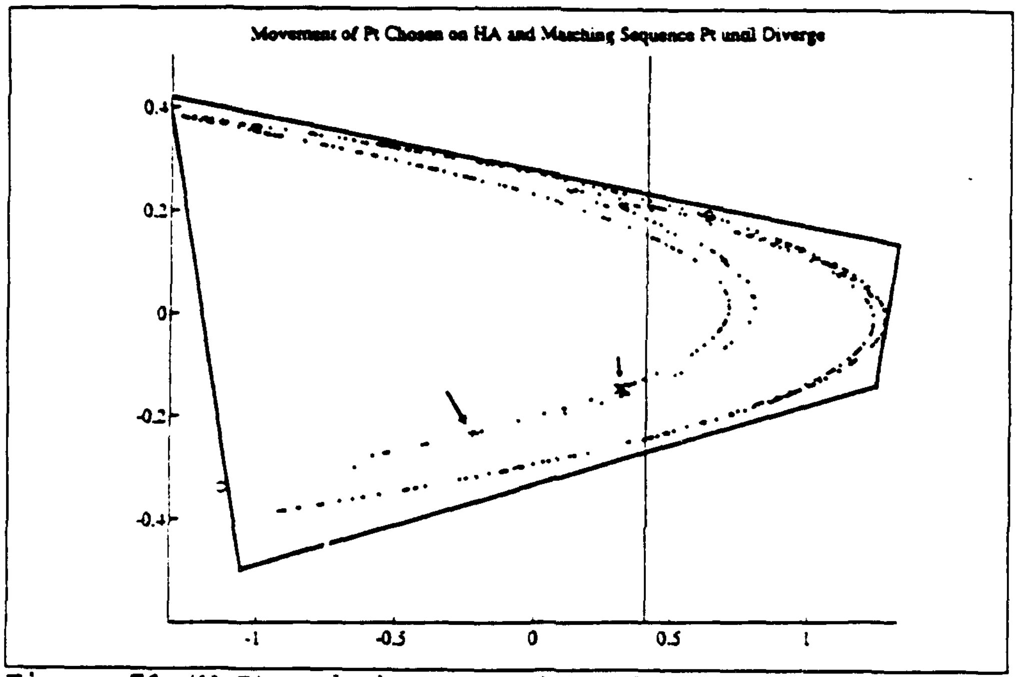

Figure 71 (W,Z) and three (u,v)'s: Iterate 4 (Notice that the (u,v) points are essentially superimposed).

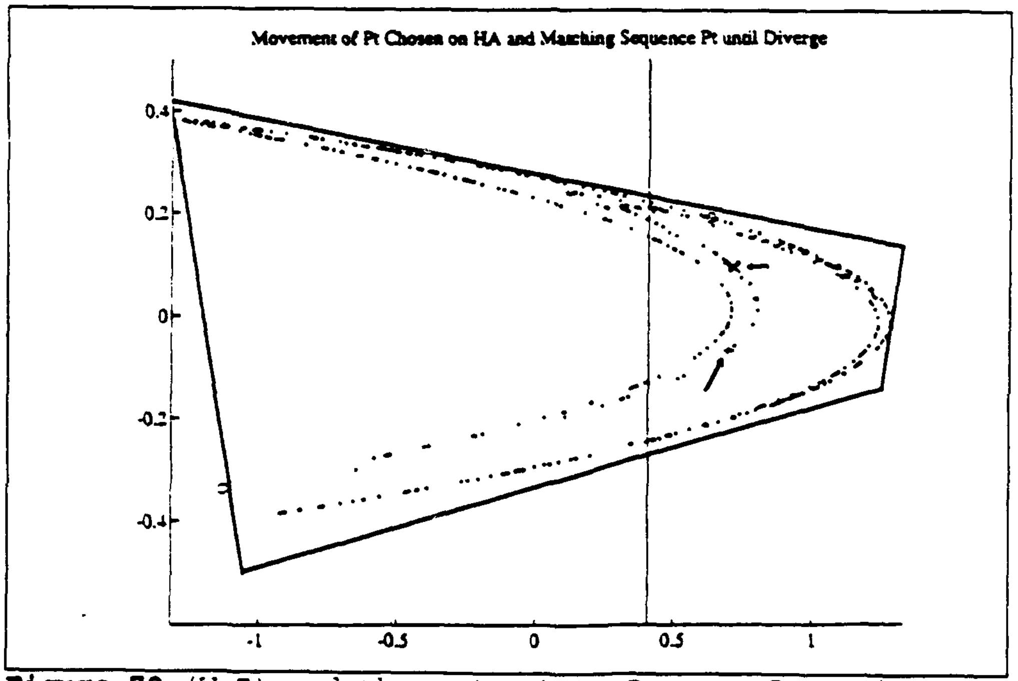

Figure 72 (W,Z) and three (u,v)'s: Iterate 5 (Notice that the (u,v) points are essentially superimposed).

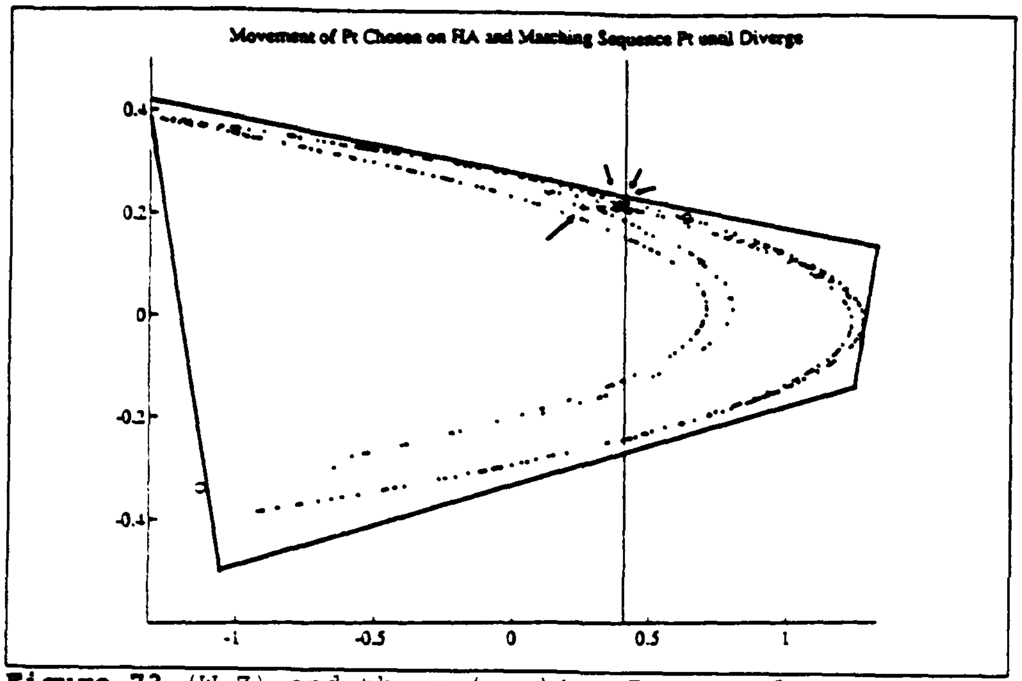

Figure 73 (W,Z) and three (u,v)'s: Iterate 6

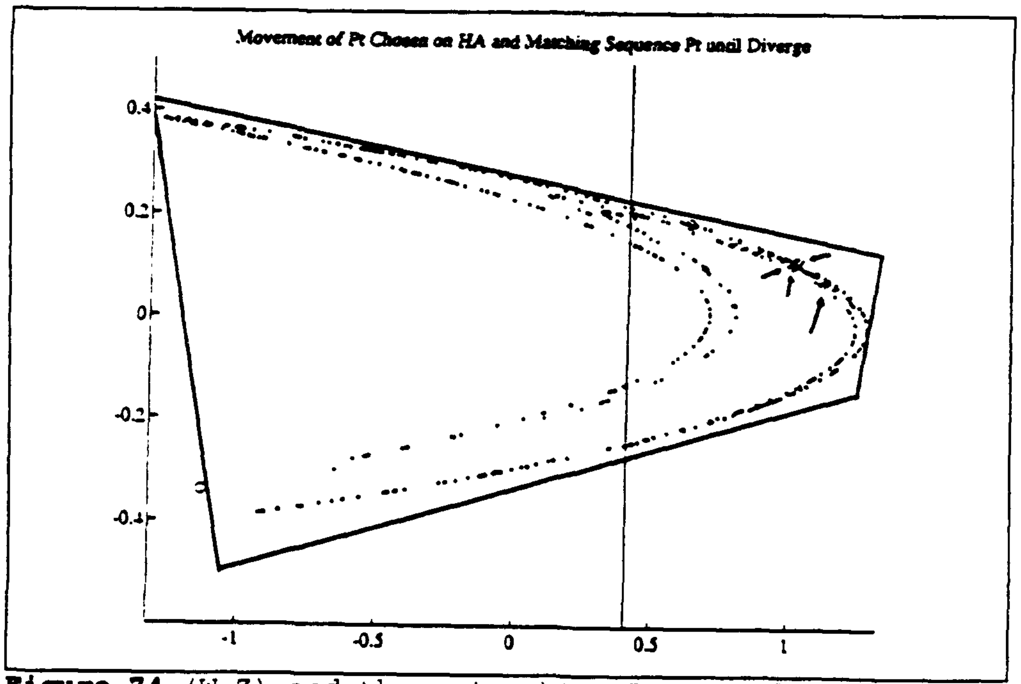

Figure 74 (W,Z) and three (u,v)'s: Iterate 7

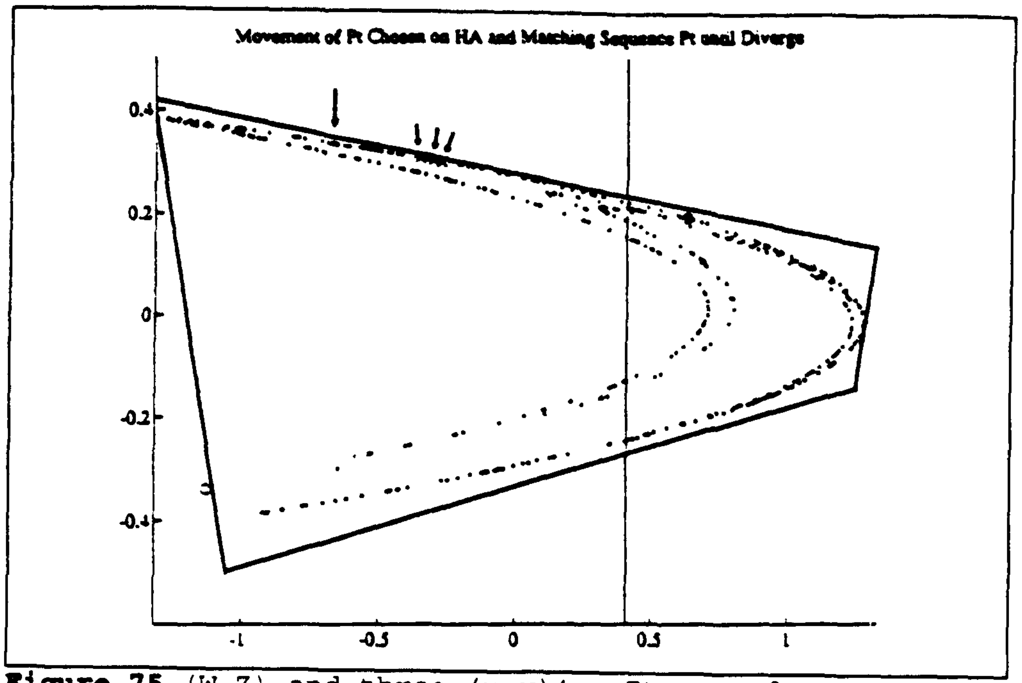

Figure 75 (W,Z) and three (u,v)'s: Iterate 8

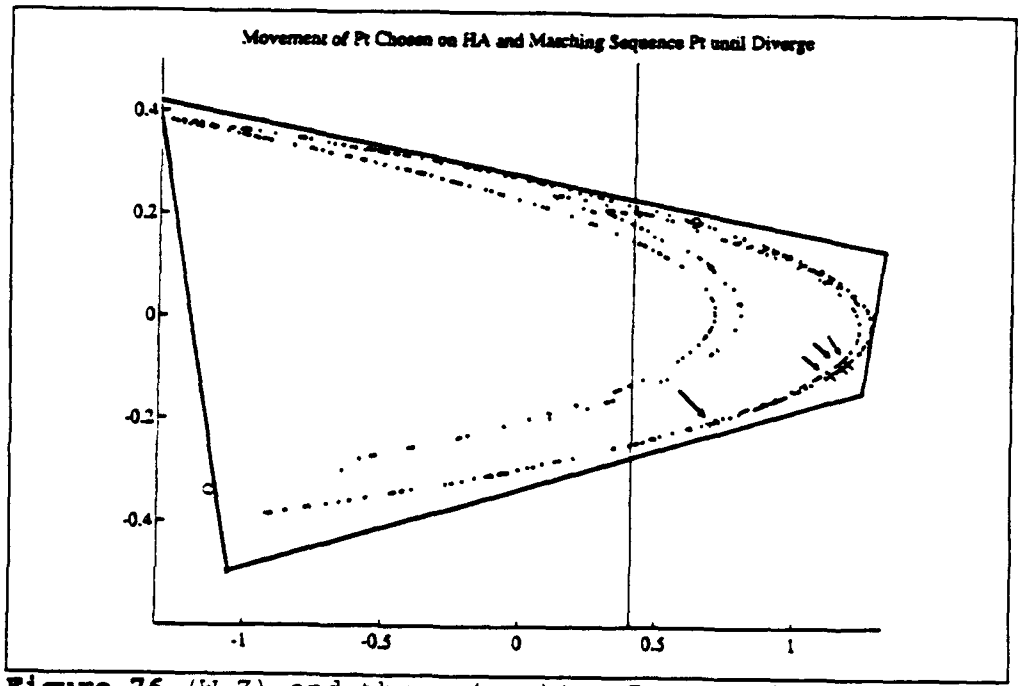

Figure 76 (W,Z) and three (u,v)'s: Iterate 9

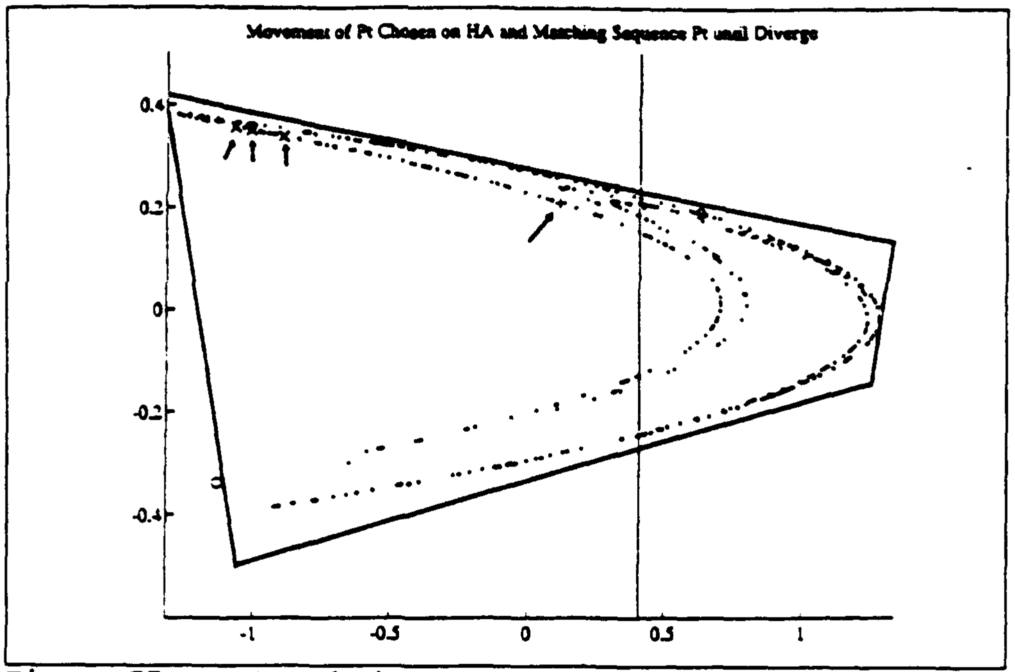

Figure 77 (W,Z) and three (u,v)'s: Iterate 10

APPENDIX C: 6 TO 16-TUPLE DATA

This appendix contains the counts of n-tuples for typical Hénon binary sequences for n-tuples of length 6-16.

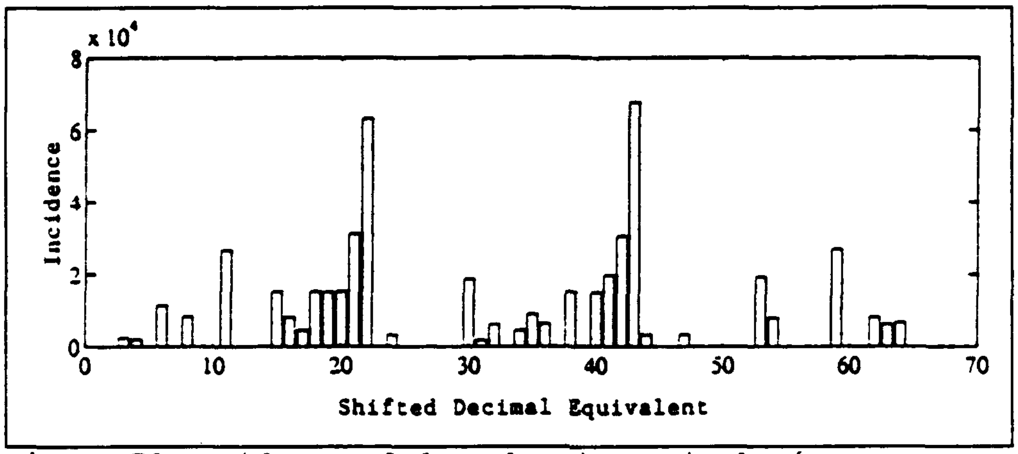

Figure 78 Incidence of 6-tuples in typical Hénon generated binary sequence

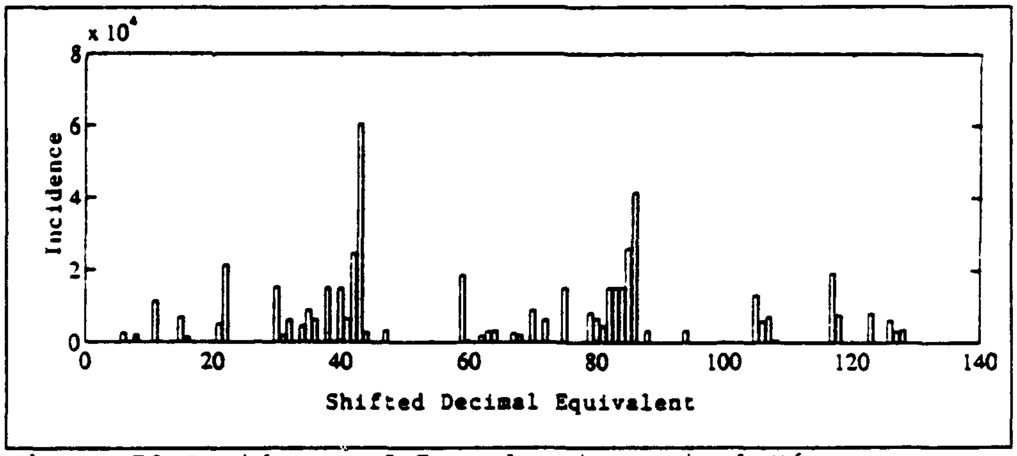

Figure 79 Incidence of 7-tuples in typical Hénon generated binary sequence

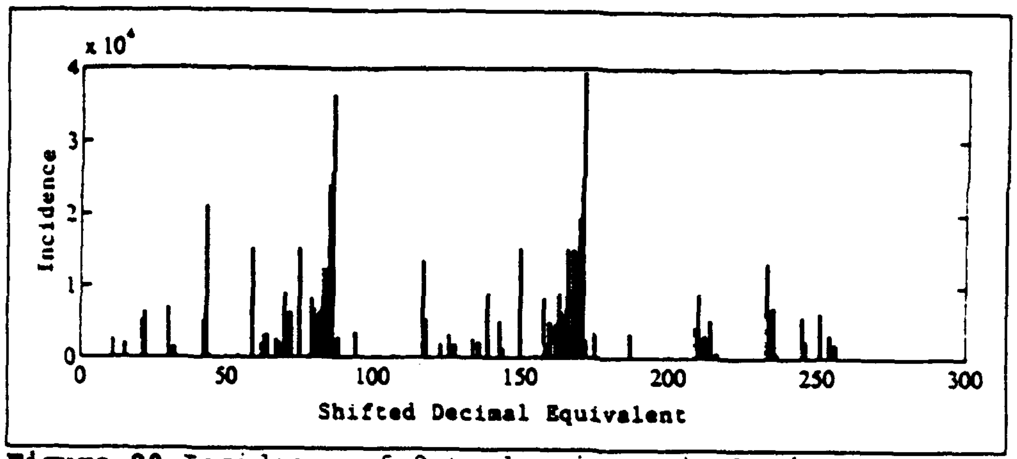

Figure 80 Incidence of 8-tuples in typical Hénon generated binary sequence

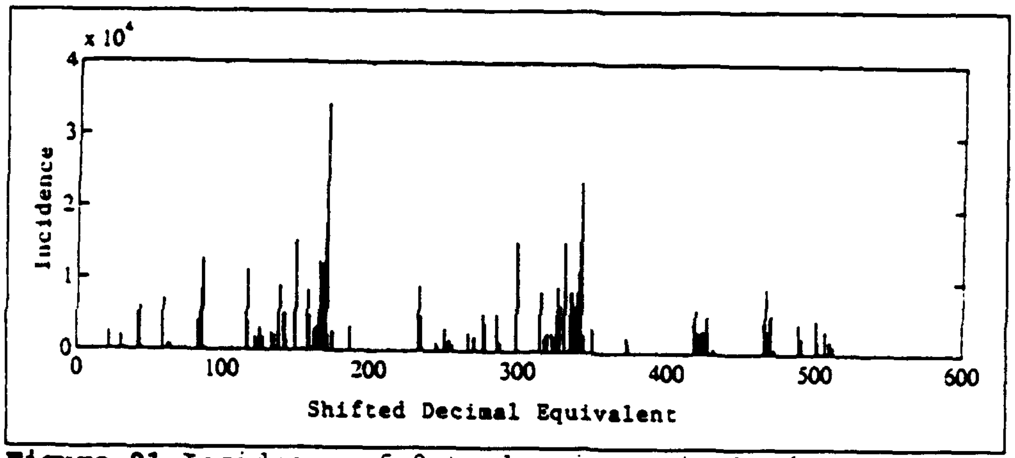

Figure 81 Incidence of 9-tuples in typical Hénon generated binary sequence

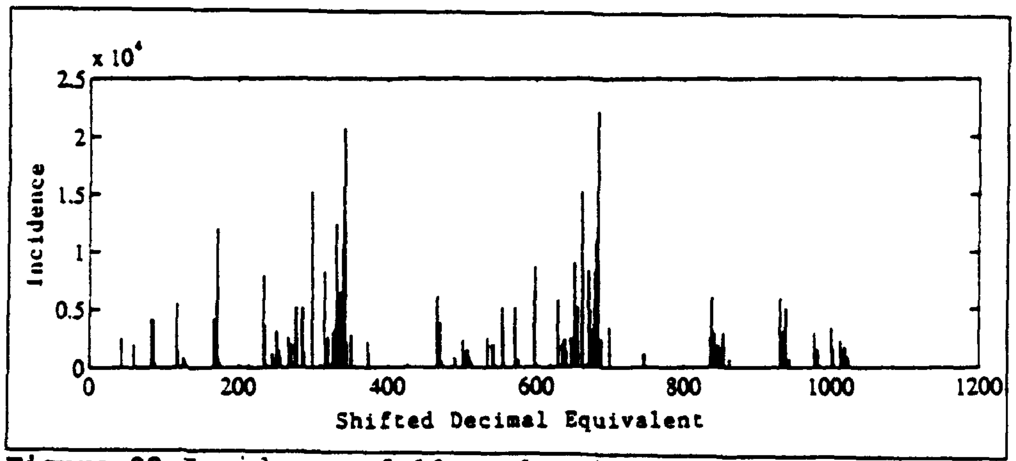

Figure 82 Incidence of 10-tuples in typical Hénon generated binary sequence

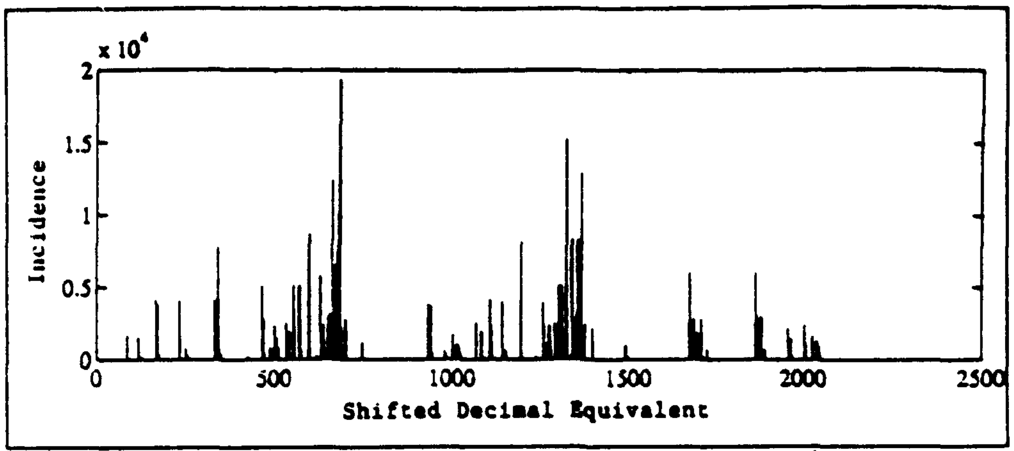

Figure 83 Incidence of 11-tuples in typical Hénon generated binary sequence

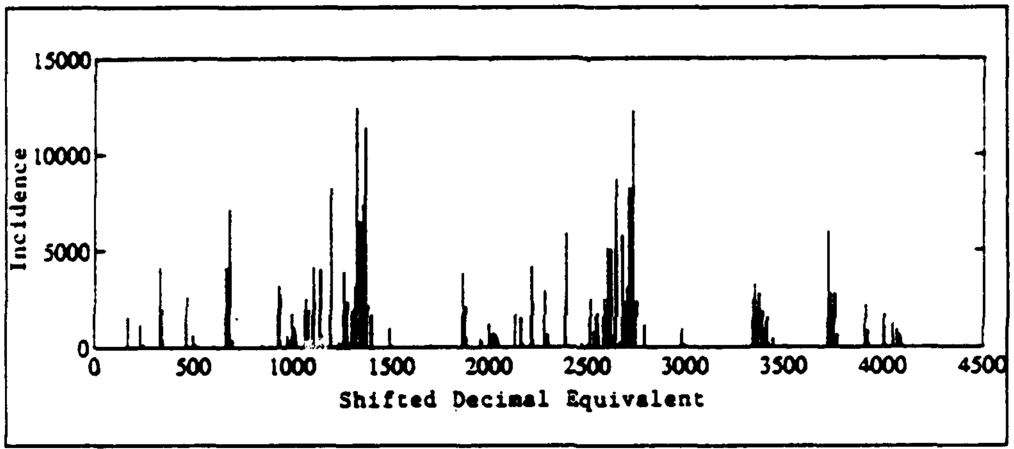

Figure 84 Incidence of 12-tuples in typical Hénon generated binary sequence

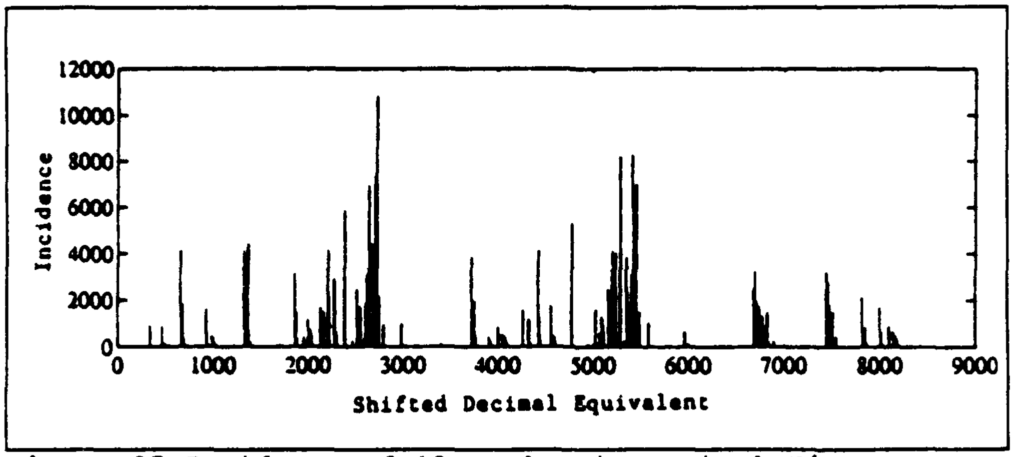

Figure 85 Incidence of 13-tuples in typical Hénon generated binary sequence

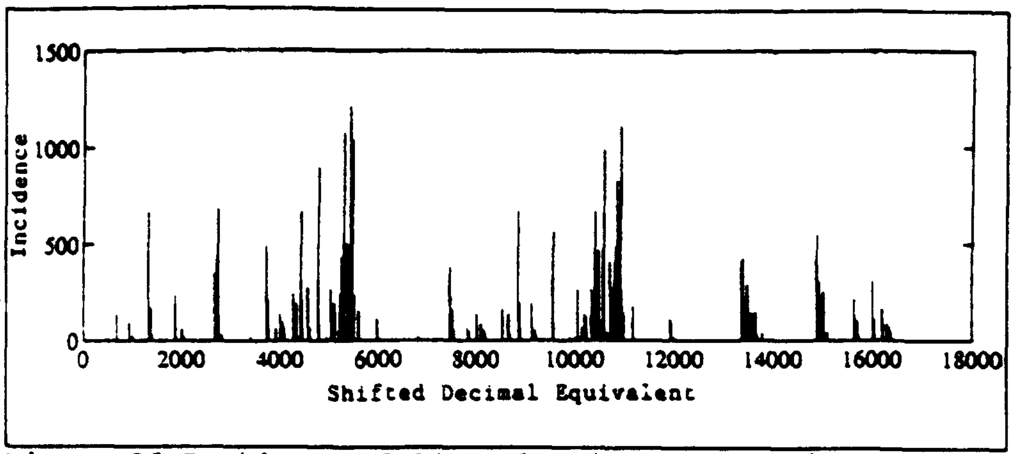

Figure 86 Incidence of 14-tuples in typical Hénon generated binary sequence

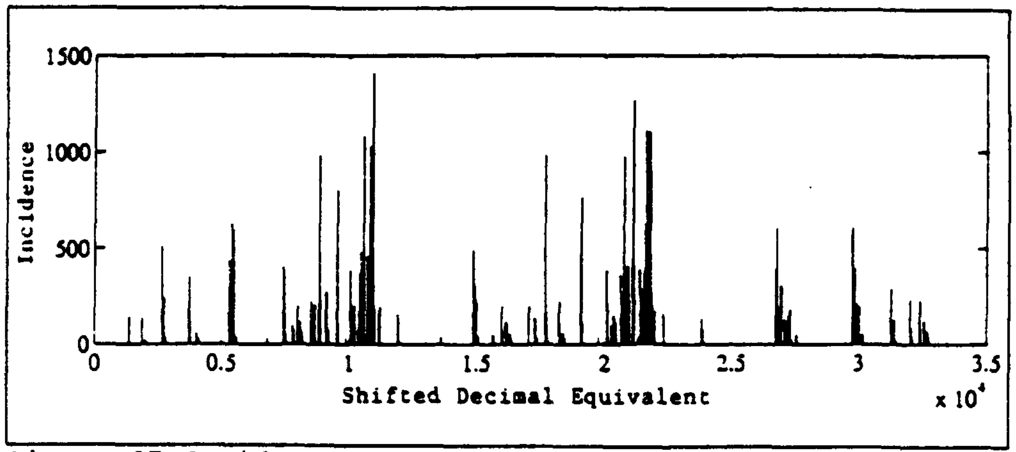

Figure 87 Incidence of 15-tuples in typical Hénon generated binary sequence

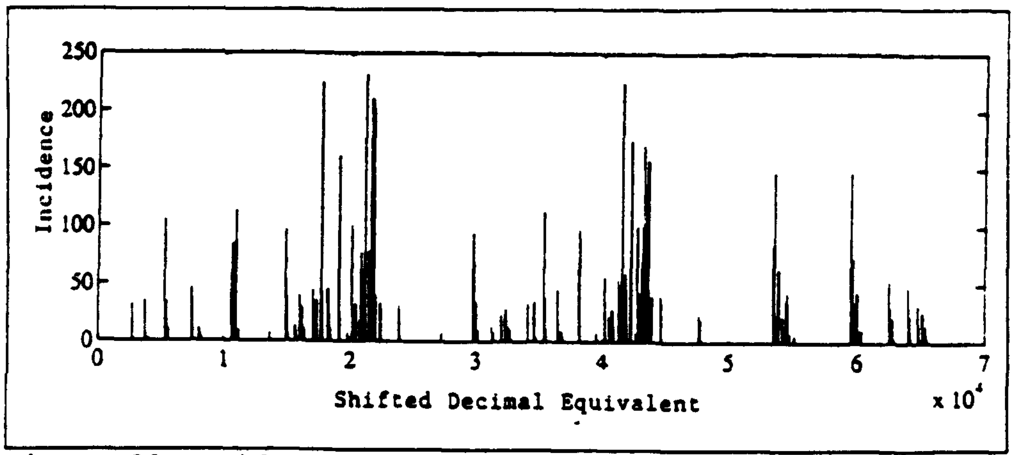

Figure 88 Incidence of 16-tuples in typical Hénon generated binary sequence

APPENDIX D: AN UNREALIZABLE 5-TUPLE

As we saw in Chapter IV, just because a particular n-tuple is not found in a typical string of length 10^5 does not mean we will not find it in a typical string of length 10^6 or longer. The only way to show that an n-tuple is unrealizable is to prove it analytically as we have done for the sequence {1,1,0,0}. It is in this Appendix that we prove that the sequence {0,0,0,0,0} is also unrealizable. This specific 5-tuple is used in Chapter V in order to support a conclusion as to *why* certain sequences are unrealizable.

The sequence {0,0,0,0,0} is one of the four 5-tuples that appeared to be unrealizable from our data in addition to those 5-tuples that included the sequence {1,1,0,0}. These 5-tuples appear in Chapter IV Table 4. Consider the 5-tuple {0,0,0,0,0}. Suppose this binary sequence corresponds to the following sequence of x values: $\{x_{n-1}, x_n, x_{n+1}, x_{n+2}, x_{n+3}\}$. All these x values must, therefore, be at most .4098. Referring to equation (1.3) on page 10 the possible solutions for x_{n+2} correspond to the region beneath the inverted parabola in Figure 89. Similarly, only those x_{n+1} and x_n terms less than or equal to .4098 are considered (refer to Figure 90).

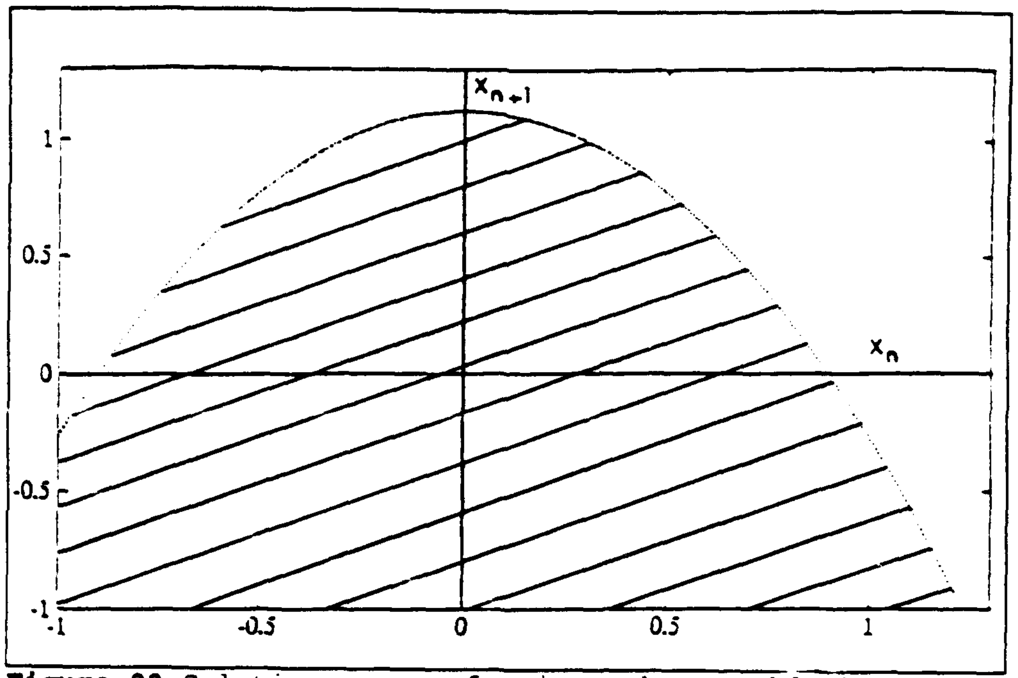

Figure 89 Solution space of points that could give the sequence {0,0,0,0,0} under x_{n-1} restriction

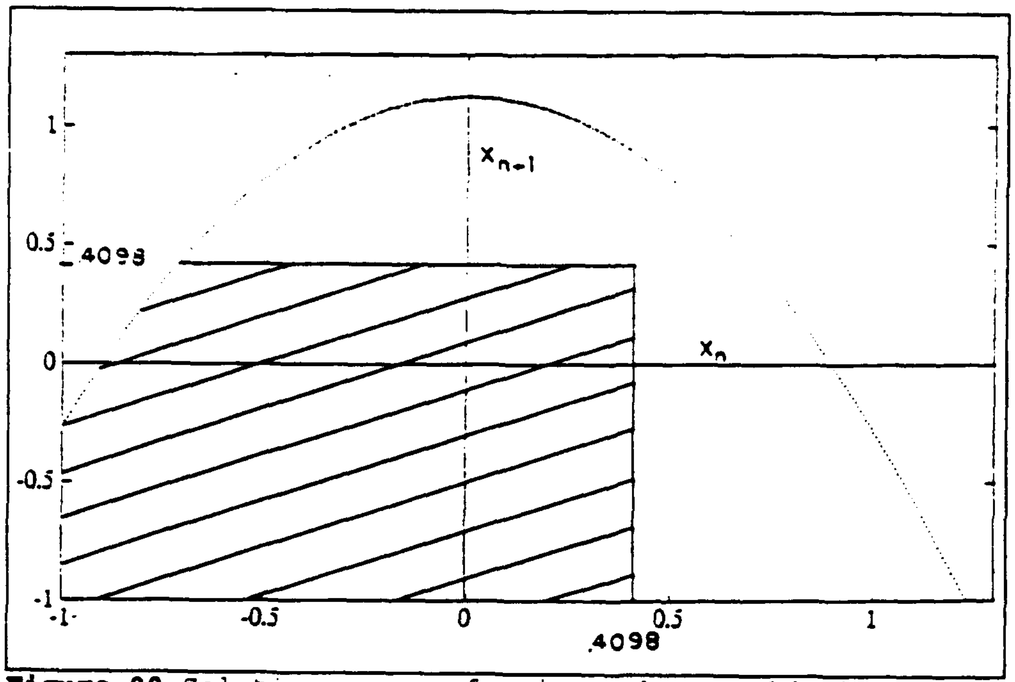

Figure 90 Solution space of points that could give the sequence {0,0,0,0,0} under x_n and x_{n+1} restrictions

By containing the trapping region in a rectangle (see Figure 91) we see how the possible solution values can be further constrained.

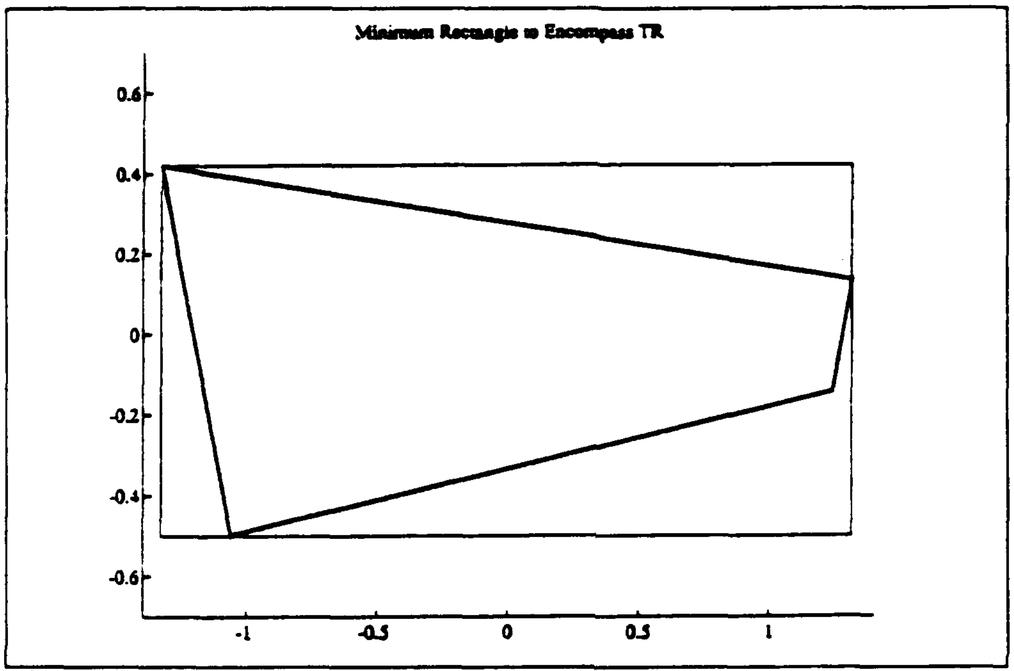

Figure 91 Trapping region in minimum area rectangle

The minimum and maximum x values, for example, are -1.32 and 1.33. In order to show that a string of five successive zeros is not possible we consider the recurrence:

$$x_{n+3} = 1 - 1.4x_{n+2}^2 + .3x_{n+1} \qquad \text{(D.1)}$$

where x_n and x_{n-1} are implicit in the equations. To optimize our potential of finding a value for x_{n+3} at most x_{MED}=.4098 we require a maximum x_{n+2} value and a minimum x_{n+1} value. Clearly,

x_{n+1} can be no less than -1.32. However, x_{n+2} is not limited only by .4098, as we shall see. Consider the equation:

$$x_{n+2} = 1 - 1.4x_{n+1}^2 + .3x_n.$$

To find our desired maximum x_{n+2} value, we must use a minimum value for x_n and a maximum value for x_{n+1}, such that

$$x_{n+2} \leq 1 - 1.4(.4098)^2 + .3(-1.32).$$

This further constrains x_{n+2} to a value no greater than approximately .3689 (see Figure 92). (The value of x_{n+2} to eight decimal places is .36888954).

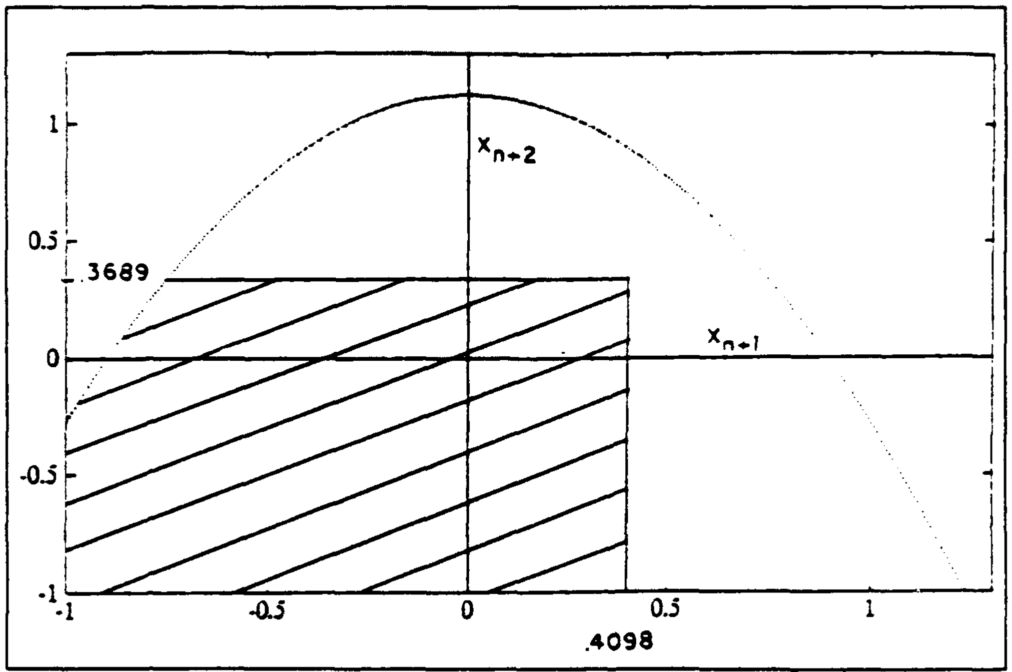

Figure 92 Solution space of points that could give the sequence {0,0,0,0,0} under x_{n+1} and x_{n+2} restrictions

Thus, referring to equation (D.1), the maximum x_{n+2} value is no more than .3689 and the minimum x_{n+1} value is -1.32 based on the confines of the trapping region. It follows that

$$x_{n+3} \geq 1 - 1.4(.3689) + .3(-1.32)$$

$$x_{n+3} \geq .41348871 \text{ (to eight decimal places).}$$

The value of x_{n+3} corresponds to a binary value of 1. Thus {0,0,0,0,0} is not realizable under this symbolic dynamics.

APPENDIX B: SUBORDINATE PROGRAMS

This Appendix includes programs which are necessary but secondary to the programs in Appendix A. Two of the programs (HENON and HENREAL) are essential for the proper operation of the APPENDIX A programs.

```
function [x,y] = henreal(n,x0,y0)
% NOTE: THIS PROGRAM IS REQ'D TO RUN ALL GRDCOMP PROGRAMS
% This program is credited in its entirety to the author of
% reference 13;it has been used by permission.
% program to generate n-length real sequences based on the %
% Hénon
% horseshoe attractor.  initial points fit into the
% quadrilateral of convergence described in [Hen76].
%inputs:
%       n = length of desired sequence
%      x0 = initial x value
%      y0 = initial y value
%outputs:
%       x = n by 1 real vector
%       y = n by 1 real vector
%
x = zeros(n,1);
y = zeros(n,1);
x(1) = x0;
y(1) = y0;
%routine to check if initial points are valid
A=[3.4074 1;-.1083 -1; -3.64 1; -.1562 1];
B=[-4.1119 -.2760 -4.6718 -.3344]';
if min((A*[x0;y0]) > B) == 0
        disp('initial point outside convergence zone')
        return
end
%recursive generation of points
for i = 1:n-1;
        x(i+1) = y(i) + 1 - 1.4*x(i)^2;
        y(i+1) = .3 * x(i);
end
%plot(x,y,'b.');
```

```
% PROBSUBQUAD.M
% PROGRAM REQUIRES HENREAL PROGRAM
% This program finds the dynamic probability of a point
falling into the
% exclusion zone compared to being on the rest of the
attractor.
[x,y] = henreal(100000,0,0);

a = [232.7 1; -.14 -1; -1.769 1; .2673 1];
b = [95.6 -.29 -.8 .25]';
%z = zeros(length(x));

for i = 1:length(x);
        if min((a*[x(i);y(i)]) > b) == 0  % if outside
subquad
                z(i) = 1;
        else    z(i) = 0;
        end
end
t  = find(z==0);
tt = length(t);

probability110 = tt/length(x)
pause;
subplot(2,1,1),plot(x(t),y(t),'w.')
```

```
% function [xmat, ymat] = grdcomp5(sp)
% ONLY DIFF BTWN 4&5 IS THAT 4 CHOOSES W,Z FROM THE LEFT
QUAD THEN ENSURES
% THIS POINT IS ON THE ATTRACTOR, 5 CHOOSES FROM THE RIGHT
QUAD AT RANDOM
% THEN ENSURES THAT THIS POINT IS ON THE ATTRACTOR.
% This function takes a point(W,Z)from the quadr and
iterates the point n
% times in order to ensure the point is on the Henon
attractor(ensuring
% that the nth iterate is in the left quadrant). It
initiates a grid field
% based on a certain spacing, then iterates those grid field
points that
% match the binary string of (wnew,znew). As long as
% the strings match, points are iterated using the Henon
recurrence. Only that
% (those) points that completely match are finally plotted
according to the
% following scheme:
%    g.    original grid field points
%    w+    original (W,Z)
%    gx    original xvec,yvec point(s) which matches binary
string of W,Z

format long
% We also show how W,Z and xvec(indices),yvec(indices)
"walk" to the Henon
% attractor at each iteration (with the attractor on
screen).
o=linspace(-1.33,1.32,500);
s=-.1083*o + .276;
u=linspace(1.32,1.245,500);
v=3.64*u -4.6718;
g=linspace(-1.06,1.245,500);
h=.1533*g -.3344;
e=linspace(-1.06,-1.33,500);
f=-3.407*e - 4.1119;
 split = .4098;
 fp1x  = .6314;    fp1y = .1894;
 fp2x  = -1.1314;  fp2y = -.3393;
a = [ -.1083 -1;-3.64 1; -.1562 1];
b = [ -.2760; -4.6718; -.3344];
c = [split:sp:1.32];
d = [-.6:sp:.5];
[x, y] = henreal(500,-1.0,-.25);
 xline = [split split];        yline = [-1.32 1.32];
  lenc = length(c);             lend = length(d);
  xvec = zeros(1,lenc*lend);    yvec = zeros(1,lenc*lend);
k=0;
for p = 1:lenc
```

```
    for r = 1:lend
        k = k + 1;
% This next section puts everything outside of the quad to
(-10,-10)
        if max((a * [c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k)=d(r);
        else
           xvec(k) = -10; yvec(k) = -10;
        end
    end
end
 xvec = reshape(xvec,lenc*lend,1);
 yvec = reshape(yvec,lenc*lend,1);
    m = find(xvec ~= -10 | yvec ~= -10);
 xvec = xvec(m);
 yvec = yvec(m);

  num = rand*length(xvec);
index = fix(num) + 1;

    w = xvec(index);
    z = yvec(index);
wattr = w;           % This preserves the values of W,Z
zattr = z;
for j = 1:40                                           %
ensures wattr,zattr is
    wattr(j+1) = 1.0 - 1.4*wattr(j)^2 + zattr(j);% on the
attractor
    zattr(j+1) = .3*wattr(j);
        if j > 20 & wattr(j+1) > split, break;
        end
end
 wnew = wattr(j+1);
 znew = zattr(j+1);
wattr = wnew;
zattr = znew;
   xx = xvec;                         % preserves the values of
xvec & yvec
   yy = yvec;

axis([-1.32 1.32 -.6 .5]);
indices = 1:length(xx);               % [1 2 3 4 . . .]
      i = 1;
   qvec = length(xx);
      q = indices;
   lenq = length(xx)
   hold on;
   plot(xvec,yvec,'g.',wattr,zattr,'w+',xline,yline,'w');

plot(o,s,'w.',u,v,'w.',e,f,'w.',g,h,'w.',fp1x,fp1y,'o',fp2x,
fp2y,'o');
```

```
   hold off;
title('Grid Field and Point Chosen on Attractor');
   print;
pause;clg;
while length(q) > 1;
     axis([-1.32 1.32 -.6 .5]);
     x0 = yy + ones(size(yy)) - 1.4*xx.^2;
     y0 = .3*xx;
     w0 = znew + 1 - 1.4*wnew.^2;
     z0 = .3*wnew;
         if w0 >= split
            qtemp = find(x0 > split | x0 == split);
         else
            qtemp = find(x0 < split);
         end;
         if length(qtemp) == 0,break;
         else q = qtemp;
         end;
      i = i + 1;           % counts iterates where at least
one point matches
   qvec = [qvec,length(q)];% plot of how # with same binary
sequence
   lenq = length(q)
indices = indices(q);    % decreases each time through loop

     xx = x0(q);         % preserves to next iterate those
matching values of
     yy = y0(q);         % x0 and y0
   wnew = w0;
   znew = z0;
pause;
hold on;
   plot(xx,yy,'g.',w0,z0,'w+',xline,yline,'w');

plot(o,s,'w.',u,v,'w.',e,f,'w.',g,h,'w.',fp1x,fp1y,'o',fp2x,
fp2y,'o');
title('Movement of Points in Grid with Same Binary
Sequence');
   print;
   pause; clg; hold off;

end
axis([1 2 3 4]); axis;
vec = (1/2).^((1:length(qvec))-1);

plot((1:length(qvec))-1,qvec,'r',(1:length(qvec))-1,qvec(1)*
vec,'b');
title('Decrease in # of Grid Points with Same Binary
Sequence vs 1/2^n')
   print;
pause;clg;
```

```
% In this way, we need the index key 'indices' to tell to
which grid point the
% surviving iterate corresponds with respect to the original
xvec  and yvec.
    axis([-1.32 1.32 -.6 .5]);
 hold on;

plot(xline,yline,'w',x,y,'b.',o,s,'w.',u,v,'w.',e,f,'w.',g,h
,'w.');hold on;
   plot(xvec(indices),yvec(indices),'gx',wattr,zattr,'w+');
   plot(fp1x,fp1y,'o',fp2x,fp2y,'o');
title('wattr,zattr & Grid Pt with Same Binary Sequence');
   print;
pause;clg;hold off;
xnew = xvec(indices);
ynew = yvec(indices); % now we show a plot of how
wattr,zattr and the point
              % that generates the same binary sequence
xnew,ynew
              % walk around the attractor
 xnewt = xnew;
 ynewt = ynew;
wattrt = wattr;
zattrt = zattr;
for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wattrt.^2 + zattrt;
    z0t = .3*wattrt;
wattrt = w0t;
zattrt = z0t;
 xnewt = x0t;
 ynewt = y0t;
axis([-1.32 1.32 -.6 .5]);
hold on;

plot(x,y,'b.',xline,yline,'w-',o,s,'w.',u,v,'w.',e,f,'w.',g,
h,'w.');
   plot(fp1x,fp1y,'o',fp2x,fp2y,'o');
   plot(x0t,y0t,'gx',w0t,z0t,'w+');
title('Movement of Pt Chosen on HA and Matching Sequence Pt
until Diverge');
  print;
pause;clg;
%plot(x0t,y0t,'ix',w0t,z0t,'i+');
end
hold off;
```

```
% function [xmat, ymat] = grdcomp6(sp)
% ONLY DIFF BTWN 4&6 IS THAT 4 USED WATTR,ZATTR 5 CHOOSES AT
RANDOM FROM
% THE LEFT QUADR THEN FINDS WATTR,ZATTR.
% This function takes a point(W,Z)from the quadr and
iterates the point n
% times in order to ensure the point is on the Henon
attractor(ensuring
% that the nth iterate is in the left quadrant). It
initiates a grid field
% based on a certain spacing, then iterates those grid field
points that
% match the binary string of (wnew,znew). As long as
% the strings match, points are iterated using the Henon
recurrence. Only that
% (those) points that completely match are finally plotted
according to the
% following scheme:
%    g.    original grid field points
%    w+    original (W,Z)
%    gx    original xvec,yvec point(s) which matches binary
string of W,Z

format long
% We also show how W,Z and xvec(indices),yvec(indices)
"walk" to the Henon
% attractor at each iteration (with the attractor on
screen).
o=linspace(-1.33,1.32,500);
s=-.1083*o + .276;
u=linspace(1.32,1.245,500);
v=3.64*u -4.6718;
g=linspace(-1.06,1.245,500);
h=.1533*g -.3344;
e=linspace(-1.06,-1.33,500);
f=-3.407*e - 4.1119;
  split = .4098;
  fp1x  = .6314;    fp1y = .1894;
  fp2x  = -1.1314;  fp2y = -.3393;
a = [3.4074 1; -.1083 -1; -.1562 1];
b = [-4.1119; -.2760;  -.3344];
c = [-1.32:sp:split];
d = [-.6:sp:.5];
[x, y] = henreal(500,-1.0,-.25);
 xline = [split split];       yline = [-1.32 1.32];
  lenc = length(c);            lend = length(d);
  xvec = zeros(1,lenc*lend);   yvec = zeros(1,lenc*lend);
k=0;
for p = 1:lenc
    for r = 1:lend
        k = k + 1;
```

```
% This next section puts everything outside of the quad to
(-10,-10)
        if max((a * [c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k)=d(r);
        else
           xvec(k) = -10; yvec(k) = -10;
        end
    end
end
 xvec = reshape(xvec,lenc*lend,1);
 yvec = reshape(yvec,lenc*lend,1);
    m = find(xvec ~= -10 | yvec ~= -10);
 xvec = xvec(m);
 yvec = yvec(m);

  num = rand*length(xvec);
index = fix(num) + 1;

    w = xvec(index);
    z = yvec(index);
wattr = w;           % This preserves the values of W,Z
zattr = z;
for j = 1:40                                            %
ensures wattr,zattr is
    wattr(j+1) = 1.0 - 1.4*wattr(j)^2 + zattr(j);% on the
attractor
    zattr(j+1) = .3*wattr(j);
        if j > 20 & wattr(j+1) < split, break;
        end
end
 wnew = wattr(j+1);
 znew = zattr(j+1);
wattr = wnew;
zattr = znew;
   xx = xvec;                    % preserves the values of
xvec & yvec
   yy = yvec;

axis([-1.32 1.32 -.6 .5]);
indices = 1:length(xx);        % [1 2 3 4 . . .]
     i = 1;
  qvec = length(xx);
     q = indices;
  lenq = length(xx)
   hold on;
   plot(xvec,yvec,'g.',wattr,zattr,'w+',xline,yline,'w');

plot(o,s,'w.',u,v,'w.',e,f,'w.',g,h,'w.',fp1x,fp1y,'o',fp2x,
fp2y,'o');
   hold off;
title('Grid Field and Point Chosen on Attractor');
```

```
   print;
pause;clg;
while length(q) > 1;
    axis([-1.32 1.32 -.6 .5]);
    x0 = yy + ones(size(yy)) - 1.4*xx.^2;
    y0 = .3*xx;
    w0 = znew + 1 - 1.4*wnew.^2;
    z0 = .3*wnew;
        if w0 >= split
           qtemp = find(x0 > split | x0 == split);
        else
           qtemp = find(x0 < split);
        end;
        if length(qtemp) == 0,break;
        else q = qtemp;
        end;
      i = i + 1;           % counts iterates where at least
one point matches
   qvec = [qvec,length(q)];% plot of how # with same binary
sequence
   lenq = length(q)
indices = indices(q);   % decreases each time through loop
     xx = x0(q);        % preserves to next iterate those
matching values of
     yy = y0(q);        % x0 and y0
   wnew = w0;
   znew = z0;
pause;
hold on;
   plot(xx,yy,'g.',w0,z0,'w+',xline,yline,'w');

plot(o,s,'w.',u,v,'w.',e,f,'w.',g,h,'w.',fp1x,fp1y,'o',fp2x,
fp2y,'o');
title('Movement of Points in Grid with Same Binary
Sequence');
  print;
   pause; clg; hold off;

end
axis([1 2 3 4]); axis;
vec = (1/2).^((1:length(qvec))-1);

plot((1:length(qvec))-1,qvec,'r',(1:length(qvec))-1,qvec(1)*
vec,'b');
title('Decrease in # of Grid Points with Same Binary
Sequence vs 1/2^n')
  print
pause;clg;
% In this way, we need the index key 'indices' to tell to
which grid point the
```

```
% surviving iterate corresponds with respect to the original
xvec  and yvec.
    axis([-1.32 1.32 -.6 .5]);
hold on;

plot(xline,yline,'w',x,y,'b.',o,s,'w.',u,v,'w.',e,f,'w.',g,h
,'w.');
   plot(xvec(indices),yvec(indices),'gx',wattr,zattr,'w+');
   plot(fp1x,fp1y,'o',fp2x,fp2y,'o');
title('wattr,zattr & Grid Pt with Same Binary Sequence');
  print;
pause;clg;
hold off;
xnew = xvec(indices);
ynew = yvec(indices); % now we show a plot of how
wattr,zattr and the point
              % that generates the same binary sequence
xnew,ynew
              % walk around the attractor
 xnewt = xnew;
 ynewt = ynew;
wattrt = wattr;
zattrt = zattr;
 for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wattrt.^2 + zattrt;
    z0t = .3*wattrt;
wattrt = w0t;
zattrt = z0t;
 xnewt = x0t;
 ynewt = y0t;
axis([-1.32 1.32 -.6 .5]);
hold on;

plot(x,y,'b.',xline,yline,'w-',o,s,'w.',u,v,'w.',e,f,'w.',g,
h,'w.');
  plot(fp1x,fp1y,'o',fp2x,fp2y,'o');
  plot(x0t,y0t,'gx',w0t,z0t,'w+');
title('Movement of Pt Chosen on HA and Matching Sequence Pt
until Diverge');
 print;
pause;clg;
plot(x0t,y0t,'ix',w0t,z0t,'i+');
end
hold off;
```

```
% function [xmat, ymat] = proof5(spcrs,incr,spfn)
% HERE WE USE SPCRS TO CHOOSE A POINT AT RANDOM THEN ENSURE
IT IS ON THE
% ATTRACTOR. IT IS THEN USED AS THE COMPARISON POINT FOR ALL
FURTHER SP VALUES
% ONLY DIFF BTWN 5&6 IS THAT 6 CHOOSES W,Z FROM THE LEFT
QUADR THEN ENSURES
% THIS POINT IS ON THE ATTRACTOR, 5 CHOOSES FROM THE RIGHT
QUADR.
% A NEW WATTR,ZATTR ARE FOUND EACH TIME SP CHANGES.

% This function takes a point(W,Z)from the quadr and
iterates the point n
% times in order to ensure the point is on the Henon
attractor(ensuring
% that the nth iterate is in the left quadrant). It
initiates a grid field
% based on a certain spacing, then iterates those grid field
points that
% match the binary string of (wnew,znew). As long as
% the strings match, points are iterated using the Henon
recurrence. Only that
% (those) points that completely match are finally
considered.
 split = .4098;
    sp = spcrs;
   a = [ -.1083 -1;-3.64 1; -.1562 1];
   b = [ -.2760; -4.6718; -.3344];
   c = [split:sp:1.32];
   d = [-.6:sp:.5];
lenc = length(c);           lend = length(d);
xvec = zeros(1,lend*lenc);  yvec = zeros(1,lend*lenc);
k=0;
for p = 1:lenc
    for r = 1:lend
                k=k+1;
        if max((a * [c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k)=d(r);
        else
                        xvec(k) = -10; yvec(k) = -10;
        end
    end
end
 xvec = reshape(xvec,lenc*lend,1);
 yvec = reshape(yvec,lenc*lend,1);
    m = find(xvec~= -10 | yvec~= -10);
 xvec = xvec(m);
 yvec = yvec(m);

  num = rand*length(xvec);
index = fix(num) + 1;
```

```
    w = xvec(index);
    z = yvec(index);
wattr = w;                    % This preserves the values of
W,Z
zattr = z;
for j = 1:40
                                         %ensures wattr,zattr is
    wattr(j+1) = 1.0 - 1.4*wattr(j)^2 + zattr(j);% on the
attractor
    zattr(j+1) = .3*wattr(j);
        if j > 20 & wattr(j+1) > split, break;
        end
end
 wnew = wattr(j+1);
 znew = zattr(j+1);
for sp = spfn:incr:spcrs,                  % MAJOR OUTER LOOP
    sp = sp
   a = [ -.1083 -1;-3.64 1; -.1562 1];
   b = [ -.2760; -4.6718; -.3344];
   c = [split:sp:1.32];
   d = [-.6:sp:.5];
lenc = length(c);           lend = length(d);
xvec = zeros(1,lend*lenc);  yvec = zeros(1,lend*lenc);
k=0;
for p = 1:lenc
    for r = 1:lend
                k=k+1;
        if max((a * [c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k)=d(r);
        else
                          xvec(k) = -10;  yvec(k) -10;
        end
    end
end
 xvec = reshape(xvec,lenc*lend,1);
 yvec = reshape(yvec,lenc*lend,1);
    m = find(xvec~= -10 | yvec~= -10);
 xvec = xvec(m);
 yvec = yvec(m);

wattr = wnew;
zattr = znew;
   xx = xvec;                           % preserves the values of
xvec & yvec
   yy = yvec;
indices = 1:length(xx);         % [1 2 3 4 . . .]
      i = 1;
   qvec = length(xx);
      q = indices;
lenqvec = length(xx);
while length(q) > 1;
```

```
    x0 = yy + ones(size(yy)) - 1.4*xx.^2;
    y0 = .3*xx;
    w0 = znew + 1 - 1.4*wnew.^2;
    z0 = .3*wnew;
        if w0 >= split
           qtemp = find(x0 > split | x0 == split);
        else
           qtemp = find(x0 < split);
        end;
        if length(qtemp) == 0,break;
        else q = qtemp;
        end;
      i = i + 1;%counts iterates where ≥ one point matches
   qvec = [qvec,length(q)];% plot of how # with same binary
sequence
   lenq = length(q);
indices = indices(q);    % decreases each time through loop
    xx = x0(q);          % preserves to next iterate those
matching values of
     yy = y0(q);          % x0 and y0
   wnew = w0;
   znew = z0;
end
% In this way, we need the index key 'indices' to tell to
which grid point the
% surviving iterate corresponds with respect to the original
xvec  and yvec.
  xnew = xvec(indices);
  ynew = yvec(indices);
 xnewt = xnew;
 ynewt = ynew;
wattrt = wattr;
zattrt = zattr;
for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wattrt.^2 + zattrt;
    z0t = .3*wattrt;
 wattrt = w0t;
 zattrt = z0t;
 xnewt  = x0t;
 ynewt  = y0t;
end
nvec  = [nvec,n];
spvec = [spvec,sp];
clear xvec; clear yvec;
end                                      % END TO MAJOR OUTER LOOP
axis;
   plot(spvec,nvec,'w*')
title('SP VS. ITERATIONS UNTIL DIVERGENCE');
```

```
% function [xmat, ymat] = proof6(spcrs,incr,spfn)
% HERE WE USE SPCRS TO FIND A POINT THEN ENSURE IT IS ON THE
ATTRACTOR.
% THIS POINT IS THEN USED AS A COMPARISON POINT FOR MANY SP
VALUES.
% ONLY DIFF BTWN 4&6 IS THAT 4 USED WATTR,ZATTR 6 CHOOSES AT
RANDOM FROM
% THE LEFT QUADR THEN FINDS WATTR,ZATTR.
% This function takes a point(W,Z)from the quadr and
iterates the point n
% times in order to ensure the point is on the Henon
attractor(ensuring
% that the nth iterate is in the left quadrant). It
initiates a grid field
% based on a certain spacing, then iterates those grid field
points that
% match the binary string of (wnew,znew). As long as
% the strings match, points are iterated using the Henon
recurrence. Only that
% (those) points that completely match are finally
considered.
 split = .4098;
    sp = spcrs;
   a = [3.4074 1; -.1083 -1; -.1562 1];
   b = [-4.1119; -.2760;  -.3344];
   c = [-1.32:sp:split];
   d = [-.6:sp:.5];
lenc = length(c);            lend = length(d);
xvec = zeros(1,lend*lenc);  yvec = zeros(1,lenc*lend);
k=0;
for p = 1:lenc
    for r = 1:lend
        k=k+1;
        if max((a * [c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k)=d(r);
        else
           xvec(k) = -10; yvec(k) -10;
        end
    end
end
 xvec = reshape(xvec,lenc*lend,1);
 yvec = reshape(yvec,lenc*lend,1);
    m = find(xvec~= -10 | yvec~= -10);
 xvec = xvec(m);
 yvec = yvec(m);

  num = rand*length(xvec);
index = fix(num) + 1;

    w = xvec(index);
    z = yvec(index);
```

```
wattr = w;            % This preserves the values of W,Z
zattr = z;
for j = 1:40
% ensures wattr,zattr is
    wattr(j+1) = 1.0 - 1.4*wattr(j)^2 + zattr(j);% on the
attractor
    zattr(j+1) = .3*wattr(j);
        if j > 20 & wattr(j+1) < split, break;
        end
end
 wnew = wattr(j+1);
 znew = zattr(j+1);
for sp = spfn:incr:spcrs,                    % MAJOR OUTER LOOP
    sp = sp
   a = [3.4074 1; -.1083 -1; -.1562 1];
   b = [-4.1119; - 2760;  -.3344];
   c = [-1.32:sp:split];
   d = [-.6:sp:.5];
lenc = length(c);            lend = length(d);
xvec = zeros(1,lend*lenc);  yvec = zeros(1,lenc*lend);
k=0;
for p = 1:lenc
    for r = 1:lend
        k=k+1;
        if max((a * [c(p);d(r)]) < b) == 0
           xvec(k) = c(p); yvec(k)=d(r);
        else
           xvec(k) = -10; yvec(k) -10;
        end
    end
end
 xvec = reshape(xvec,lenc*lend,1);
 yvec = reshape(yvec,lenc*lend,1);
    m = find(xvec~= -10 | yvec~= -10);
 xvec = xvec(m);
 yvec = yvec(m);
wattr = wnew;
zattr = znew;
   xx = xvec;                           % preserves the
values of xvec & yvec
   yy = yvec;
indices = 1:length(xx);         % [1 2 3 4 . . .]
      i = 1;
   qvec = length(xx);
      q = indices;
lenqvec = length(xx);
while length(q) > 1;
    x0 = yy + ones(size(yy)) - 1.4*xx.^2;
    y0 = .3*xx;
    w0 = znew + 1 - 1.4*wnew.^2;
    z0 = .3*wnew;
```

```
         if w0 >= split
            qtemp = find(x0 > split | x0 == split);
         else
            qtemp = find(x0 < split);
         end;
         if length(qtemp) == 0,break;
         else q = qtemp;
         end;
      i = i + 1;                  % counts iterates where at
least one point matches
   qvec = [qvec,length(q)]; % plot of how # with same binary
sequence
   lenq = length(q);
indices = indices(q);     % decreases each time through loop
    xx = x0(q);           % preserves to next iterate those
matching values of
     yy = y0(q);           % x0 and y0
   wnew = w0;
   znew = z0;
end
% In this way, we need the index key 'indices' to tell to
which grid point the
% surviving iterate corresponds with respect to the original
xvec  and yvec.
  xnew = xvec(indices);
  ynew = yvec(indices);
 xnewt = xnew;
 ynewt = ynew;
wattrt = wattr;
zattrt = zattr;
for n = 1:i - 1
    x0t = 1.0 - 1.4*xnewt.^2 + ynewt;
    y0t = .3*xnewt;
    w0t = 1.0 - 1.4*wattrt.^2 + zattrt;
    z0t = .3*wattrt;
 wattrt = w0t;
 zattrt = z0t;
 xnewt  = x0t;
 ynewt  = y0t;
end
nvec  = [nvec,n];
spvec = [spvec,sp];
clear xvec;clear yvec;
end                               % END TO MAJOR LOOP
axis;
   plot(spvec,nvec,'w*')
title('SP VS. ITERATIONS UNTIL DIVERGENCE');
```

```
function x = henon(n,x0,y0)
% This program is credited to the author of reference 13; it
% has been used by permission.
%function x = henon(n,x0,y0)
%program to generate n-length binary sequences based on the
%Hénon
%horseshoee attractor.  initial points are checked %against
the
%quadrilateral of convergence. (Hénon 1976)
%inputs:
%       n = length of desired sequence
%      x0 = initial x value
%      y0 = initial y value

%outputs:
%       x = n by 1 binary vector
%
x(1) = x0;
y(1) = y0;
split = .4098;   %median x-value of henon attractor

%routine to check if initial points are valid
A=[3.4074 1;-.1083 -1; -3.64 1; -.1562 1];
B=[-4.1119 -.2760 -4.6718 -.3344]';
if min((A*[x0;y0]) > B) == 0
        disp('initial point outside convergence zone')
        return
end

x(2:n) = zeros(n-1,1);   %vectors are preallocated here to

                    %save time
y(2:n) = zeros(n-1,1);   %in case initial point is outside

                    %of zone

%recursive generation of points
for i = 1:n;
        x(i+1) = y(i) + 1 - 1.4*x(i)^2;
        y(i+1) = .3 * x(i);

%convert previous point to binary
        if x(i) <= split
                x(i) = 0;
        else
                x(i) = 1;
        end
end
%minor housekeeping to dump the last term
x = x(1:n);
```

LIST OF REFERENCES

1. Gulick,D., *Encounters With Chaos*, McGraw-Hill, Inc., 1992.

2. Holden,A.V., *Chaos*, p.19, Princeton University Press, 1986.

3. Barnsley,M.F., and Demko,S.G., *Chaotic Dynamics and Fractals*, v.2, p.102, Academic Press, Inc., 1986.

4. Cipra,B.A., "Two Mathematicians Prove A Strange Theorem", *SIAM* News, v.24, no.3, pp.1,7,9, May 1991.

5. Peitgen,H., Jurgens,H., and Saupe,D., *Chaos and Fractals: New Frontiers of Science*, Springer-Verlag, 1992.

6. Stewart,I., *Does God Play Dice? The Mathematics of Chaos*, p.162, Basil Blackwell Inc., 1992.

7. Hénon,M., "A two dimensional mapping with a strange attractor", *Communications in Mathematical Physics*, v.50, pp.69-77, 1976.

8. Devaney,R.L., *An Introduction to Chaotic Dynamical Systems*, Second Edition, Addison-Wesley Publishing Co, Inc., 1989.

9. Guckenheimer,J., and Holmes,P., *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*, p.251, Springer-Verlag New York Inc., 1983.

10. Arrowsmith,D.K., and Place,C.M., *An Introduction to Dynamical Systems*, p.2,351-352, Cambridge University Press, 1991.

11. Bowen,R., *On Axiom A Diffeomorphisms. Conference Board of the Mathematical Sciences Regional Conference Series In Mathematics*, v.35, AMS Publications, 1978.

12. Forré,R., "The Hénon Attractor as a Keystream Generator", preprint, 1990.

13. Heyman,J.E., *On The Use of Chaotic Dynamical Systems to Generate Pseudorandom Bitstreams*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1993.

14. Benedicks,M., and Carleson, L., *The Dynamics of the Hénon Map*, Annals of Mathematics, v.133, pp.73-169, 1991.

15. Chow,S. and Palmer,K.J., "The Accuracy of Numerically Computed Orbits of Dynamical Systems in R^k", *Journal of Dynamics and Differential Equations*, v.3, no.3, pp.39-45, 1991.

16. Naval Postgraduate School Report NPS-MA-93-015, *Chaotic Keystream Generators for Additive Stream Ciphers*, by Jeffery J. Leader, 6 May 1993.

17. Professor Shui-Nee Chow, Georgia Institute of Technology, Subject: Requirement for Hyperbolic Set in Hénon Attractor using Classical Parameters, 171400ZMay93.

18. Naval Postgraduate School Report NPS-MA-93-016, Neural Network Identification of Keystream Generators, by James Heyman and Jeffery J. Leader, 24 May 1993.

19. Beaver,P.F., *Fractals and Chaos*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1991.

20. Bernhard,M.A., *Introduction to Chaotic Dynamical Systems*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1991.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria VA 22304-6145	2
2.	Library, Code 052 Naval Postgraduate School Monterey CA 93943-5002	2
3.	Professor R. Franke, Code MA/Fe Department of Mathematics Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Professor J. Leader, Code MA/Le Department of Mathematics Naval Postgraduate School Monterey, CA 93943-5000	5
5.	Professor I. Fischer Department of Mathematics Van Vleck Hall University of Wisconsin Madison, WI 53706	3
6.	LT Antonio Fontana 1290 5TH Street APT 1 Monterey, CA 93940	9
7.	LT James Heyman PSC 825 P.O. Box 58 FPO AE 09627	1
8.	Professor M. Stamp Mathematical Sciences Worcester Polytechnic Institute 100 Institute Rd. Worcester, MA 01609-2280	1
9.	LT Marion Holmes 3306 Del Monte Blvd. Apt. 62 Marina,CA 93933	1